ПРОЕКТУВАННЯ ПРОГРАМНИХ ДОДАНКІВ

ЧАСТИНА ІІ

САМОСТІЙНА РОБОТА СТУДЕНТІВ ТА ВИКОНАННЯ СЕМЕСТРОВИХ ЗАВДАНЬ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ПРОЕКТУВАННЯ ПРОГРАМНИХ ДОДАНКІВ

ЧАСТИНА ІІ

САМОСТІЙНА РОБОТА СТУДЕНТІВ ТА ВИКОНАННЯ СЕМЕСТРОВИХ ЗАВДАНЬ

*Рекомендовано Методичною радою НТУУ «КПІ ім. Ігоря Сікорського»
як навчальний посібник для студентів,
які навчаються за спеціальністю 151 «Автоматизація та комп'ютерно-інтегровані технології»*

Київ
КПІ ім. Ігоря Сікорського
2018

ПРОЕКТУВАННЯ ПРОГРАМНИХ ДОДАНКІВ: ЧАСТИНА ІІ. САМОСТІЙНА РОБОТА СТУДЕНТІВ ТА ВИКОНАННЯ СЕМЕСТРОВИХ ЗАВДАНЬ [Електронний ресурс]: навч. посіб. для студ. спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології» / КПІ ім. Ігоря Сікорського; уклад.: В. І. Бендюг, Б. М. Комариста. – Електронні текстові данні (1 файл: 2,87 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2018. – 215 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 10 від 21.06.2018 р.)

Електронне мережне навчальне видання

ПРОЕКТУВАННЯ ПРОГРАМНИХ ДОДАНКІВ

ЧАСТИНА ІІ

САМОСТІЙНА РОБОТА СТУДЕНТІВ ТА ВИКОНАННЯ СЕМЕСТРОВИХ ЗАВДАНЬ

Укладачі: Бендюг Владислав Іванович, канд. техн. наук, доцент
Комариста Богдана Миколаївна, канд. техн. наук

Відповідальний редактор О. О. Квітка, канд. хім. наук, доцент

Рецензенти: Ю.С. Мірошниченко, канд.техн.наук, доцент

ЗМІСТ

ВСТУП

«Прикладне програмне забезпечення - 3. Проектування програмних доданків» дисципліни «Прикладне програмне забезпечення» викладається студентам четвертого року підготовки ОКР «бакалавр» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» у восьмому навчальному семестрі. Матеріал кредитного модуля базується на дисциплінах «Комп'ютерні технології та програмування - 1. Основи алгоритмізації», «Комп'ютерні технології та програмування - 2. Програмування типових задач», «Комп'ютерна техніка та організація обчислювальних робіт», «Комп'ютерні технології та програмування - 3. Розробка інтерфейсу користувача», «Інформаційні системи та комплекси», «Прикладне програмне забезпечення - 2. Технології об'єктно-орієнтованого програмування». Знання уміння та навички, одержані під час вивчення даного модуля, у подальшому використовуються в усіх дисциплінах, які потребують програмної реалізації розрахунків на комп'ютері, і в першу чергу – в курсах «Ідентифікація та моделювання об'єктів автоматизації», «Основи проектування систем автоматизації і систем керування експериментом», «Методи штучного інтелекту та їх застосування в хімічній технології», в курсових і дипломних роботах і проектах.

Метою навчальної дисципліни є формування у студентів здібностей:

- ПК-2 - Здатність розробляти, проектувати та вдосконалювати елементи комп’ютерно-інтегрованих систем управління об’єктами і процесами, алгоритми їх функціонування.
- ПК-3 - Здатність використовувати сучасні методи і засоби проектування в розробці алгоритмічного та програмного забезпечення систем автоматизації та комп’ютерно-інтегрованих технологій.
- ПК-5 - Здатність розробляти, налагоджувати та вдосконалювати програмне забезпечення комп’ютерно-інтегрованих систем.
- ПК-11 - Здатність застосовувати комп’ютерну техніку та розробляти прикладні програмні продукти для вирішення виробничо-технічних задач.
- ПК-12 - Здатність розробляти програмно-алгоритмічні засоби реалізації методів управління в автоматизованих системах та комп’ютерно-інтегрованих технологіях.

Згідно з вимогами освітньо-професійної програми студенти після засвоєння навчальної дисципліни мають продемонструвати такі результати навчання:

знання:

- стандартів сучасних об’єктно-орієнтованих мови програмування С++;
- Інтегрованого середовища розробки програмних додатків Microsoft Visual Studio;
- сучасних методів конструювання програм засобами об'єктно-орієнтованої мови програмування С++;
- ефективних методів збереження і обробки інформації;

уміння:

- самостійно розробляти ефективні алгоритми і програми на сучасній алгоритмічній мові для вирішення поставленої задачі;
- налагоджувати програми на ПК до надійної працездатності;
- створювати додатки з використанням технології Windows Form;

досвід:

- виконувати налаштування середовища програмування для ефективної роботи;
- оформляти розроблені програми згідно вимогам ЄСПД.

1 ЗМІСТ ТА СТРУКТУРА КРЕДИТНОГО МОДУЛЯ

1.1 Структура кредитного модуля

Таблиця 1.1 – Структура кредитного модуля дисципліни

Галузь знань, напрям підготовки, освітньо-кваліфікаційний рівень	**Загальні показники**	**Характеристика кредитного модуля**
Галузь знань *0502 – Автоматика і управління* *(шифр і назва)* *Спеціальність* - *(шифр і назва)* *Спеціалізація* - *(назва)* *Освітньо-кваліфікаційний рівень* *бакалавр*	Назва дисципліни, до якої належить кредитний модуль *Прикладне програмне забезпечення*	Форма навчання денна (денна / заочна)
	Кількість кредитів ECTS 2	Статус кредитного модуля за вибором ВНЗ (нормативний або за вибором ВНЗ/студентів)
	Кількість розділів 1	Цикл до якого належить кредитний модуль ІІ.1. Дисципліни самостійного вибору навчального закладу
	Індивідуальне завдання: РГР (вид)	Рік підготовки 4
		Семестр *восьмий*
	Загальна кількість годин 60	Лекції 10 год.
		Практичні (семінарські) -
		Комп'ютерний практикум 30 год.
	Тижневих годин: аудиторних – 2,2 СРС – 1,1	Самостійна робота 20 год. в тому числі на виконання індивідуального завдання 8 год.
		Вид та форма семестрового контролю диф. залік (екзамен / залік / диф. залік; усний / письмовий / тестування тощо)

1.2. Рейтингова система оцінювання результатів навчання

Рейтинг студента з дисципліни складається з балів, що він отримує за:

1) Виконання та захист комп'ютерних практичних робіт;
2) Виконання розрахунково-графічної роботи.

Шкала балів за відповідні рівні оцінювання з кожного виду контролю. З урахуванням межових значень 0,9 – 0,75 – 0,6 – 0 маємо такий розподіл:

а) Комп'ютерні практичні роботи:

«відмінно» –9-10 балів;

«добре» – 7-8 балів;

«задовільно» – 6 балів;

«незадовільно» – 0 балів.

б) РГР:

«відмінно» – 27-30 балів;

«добре» – 23-26 балів;

«задовільно» – 18-22 балів;

«незадовільно» – 0 балів.

Контрольна перевірка: студент, який отримав мінімальні позитивні бали за всіма контролями, матиме у підсумку не менше 60 балів.

$$6 \times 7 + 18 = 60 \text{ балів.}$$

Система рейтингових (вагових) балів та критерії оцінювання

1. Комп'ютерні практичні роботи.

Ваговий бал – 10.

Рейтингові бали комп'ютерної практичної роботи складаються з балів за підготовку та виконання роботи (від 1 до 2), балів за оформлення протоколу роботи (від 1 до 2 балів) і балів за захист роботи (від 4 до 6). Таким чином за результатами роботи студент може отримати від 6 до 10 балів.

За ***виконання роботи*** бали виставляються наступним чином:

- робота повністю і вірно виконана у відведений час – 2 бали;
- робота виконана у відведений час, але не повністю висвітлює деякі аспекти завдання – 1 бал;
- робота виконана невчасно або має суттєві недоліки – 0 балів.

За ***оформлення протоколу роботи*** бали виставляються наступним чином:

- протокол відповідає вимогам, оформлений охайно, без виправлень і помарок (допускається не більше 1 виправлення на 1 сторінці протоколу) – 2 бали;
- протокол відповідає вимогам, проте є певні недоліки – 1 бал;
- протокол не відповідає основним вимогам до оформлення – 0 балів.

За ***захист роботи*** бали виставляються наступним чином:

- студент вірно і повністю відповів на всі поставлені йому запитання (виконав надані для захисту роботи завдання) – 6 балів;
- студент відповів на запитання в цілому вірно, проте при відповідях припускався несуттєвих помилок – 4-5 балів;
- студент припустився помилок при відповідях на більшість питань або взагалі не відповів на більшу частину запитань – (0 балів);
- при отриманні 0 балів за захист, студент має повторно захищати лабораторну роботу на наступному занятті.

2. Розрахунково-графічна робота

Ваговий бал – 30.

Розрахунково-графічна робота являє собою практичне завдання – створення програмного модулю на мові програмування С++ згідно отриманого варіанту індивідуального завдання. Оцінювання такої роботі проводиться за наступною шкалою:

- студент повністю вірно і у повному обсязі виконав поставлене завдання та відповів на додаткові питання (27-30 балів);
- студент повністю вірно і у повному обсязі виконав поставлене завдання але не відповів на додаткові питання (від 23 до 26 балів);
- студент в цілому вірно та повністю виконав поставлене завдання але припустився несуттєвих помилок (від 18 до 22 балів);
- студент виконав завдання неповністю та (або) припустився при виконанні суттєвих помилок (0 балів);
- студент не здав розрахунково-графічну роботу у відведений час без поважної причини (– 1 бал за кожен тиждень затримки).

Штрафні та заохочувальні бали за *:

- Вдосконалення дидактичного матеріалу до комп'ютерної практичної роботи +1 бал;
- Вдосконалення дидактичного матеріалу до розрахунково-графічної роботи +5 балів;
- Недопущення до комп'ютерної практичної роботи у зв'язку з незадовільним вхідним контролем (відсутність протоколу, недостатня підготовка до роботи) –1 бал;
- Відсутність на лабораторному занятті без поважної причини –1 бал;
- Відсутність у студента конспекту під час перевірки на лекційному занятті –1 бал;

Умови допуску до заліку

Умовами допуску до заліку є:

- Захист 7-ми комп'ютерних практичних робіт на позитивну оцінку (6 і більше балів);
- Виконання розрахунково-графічної роботи на позитивну оцінку (18 або більше балів);
- Наявність конспекту лекцій.

Умови отримання семестрової атестації

Календарна атестація студентів (на 8 та 14 тижнях семестрів) з дисциплін проводиться викладачами за значенням поточного рейтингу студента на час атестації. Якщо значення цього рейтингу не менше 50 % від максимально можливого на час атестації, студент вважається задовільно атестованим. В іншому випадку в атестаційній відомості виставляється «незадовільно».

1-ша атестація

$$r_C = (3*10)*0,5=15$$

Якщо $r_C \geq 15$ студент вважається задовільно атестованим

2-га атестація

$$r_C = (6*10)*0,5=30$$

Якщо $r_C \geq 30$ студент вважається задовільно атестованим

Розрахунок шкали рейтингу:

Сума вагових балів контрольних заходів протягом семестру складає:

$$R_C = 10*7 + 30 = 100 \text{ балів.}$$

Таким чином, рейтингова шкала з дисципліни складає **R = 100 балів**.

Студенти, які набрали протягом семестру рейтинг менше **60 балів** зобов'язані виконувати залікову контрольну роботу.

Студенти, які набрали протягом семестру необхідну кількість балів (**60 балів і більше**), мають можливості отримати залікову оцінку (залік) відповідно до набраного рейтингу (табл. 1);

Студенти, які наприкінці семестру мають рейтинг менше 60 балів, а також ті, хто хоче підвищити оцінку в системі ECTS, виконують залікову контрольну роботу. При цьому до балів за РГР ($r_{РГР}$) додаються бали за контрольну роботу і ця рейтингова оцінка є остаточною.

Завдання контрольної роботи складається з *практичного завдання по створенню програмного комплексу на мові С++. Додаткове питання* з тем лекційних занять отримують студенти, які були відсутні на певній лекції. Незадовільна відповідь з додаткового питання знижує загальну оцінку на 2 бали.

Оцінка за залікову контрольну роботу складається з 3-х частин: вірність і оптимальність розрахункового алгоритму програми – 25 балів; обробка та введення вихідних даних і виведення результатів розрахунку – 25 балів; оформлення інтерфейсу користувача, його зручність та оптимальність – 20 балів.

Оцінювання такої роботи проводиться за наступною шкалою:

- програмний комплекс відповідає всім вимогам та виконує вірно всі розрахунки – 63-70 бали;
- програмний комплекс працює вірно, проте, не відповідає певним вимогам до зручності програмного інтерфейсу – 53-62 балів;
- програмний комплекс працює в цілому вірно, проте, не відповідає більшості вимог до створення програмного інтерфейсу, або обробки вхідних даних та виведення результатів розрахунку – 42-52 балів;

- програмний комплекс працює не вірно, інтерфейс не відповідає сучасним вимогам, обробка вхідних даних та результатів не на належному рівні – 0 балів;
- якщо в програмному комплексі присутні помилки, або інтерфейс повною мірою не відповідає вимогам, то бали знімаються наступним чином:
  - програмний комплекс працює, але не вірно виконує розрахунки – знімається 5 балів;
  - не запрограмоване обробник помилок або не всі стандартні помилки при роботі програми перехоплюються програмою чи не виводяться відповідні конкретні повідомлення по певним типам помилок – знімається 3 бали;
  - відсутні підказки для користувача при роботі з програмою – знімається 3 бали;
  - не передбачена можливість завантаження даних з файлу чи збереження результатів роботи у файл за допомогою відповідних засобів мови С++ – знімається 3 бали;
  - не оптимально виконана компоновка програмного коду, не виділені блоки введення даних і виведення результатів окремими логічними блоками - знімається 2 бали;
  - допущено дрібні помилки при створенні чи роботі програмного комплексу – знімається 1 бал за кожну помилку.

В разі, якщо студент не закінчив виконання роботи вчасно, оцінюється та частина, яка фактично виконана.

Загальна сума балів за залікову контрольну роботу розподіляється наступним чином:

- ***«відмінно»,*** завдання виконане повністю без помилок (не менше 90% потрібної інформації) – ***63-70 балів***;
- ***«добре»,*** завдання виконане в достатньому обсязі з дрібними неточностями (не менше 75% обсягу завдання або незначні неточності) – ***53-62 бали***;
- ***«задовільно»,*** виконана більша частина завдання, або наявні деякі помилки (не менше 60% виконаного завдання та деякі помилки) – ***42-52 бали***;
- ***«незадовільно»,*** незадовільне виконання завдання – ***0 балів***.

$$R = r_{РГР} + r_{ЗКР}$$

Сума балів за виконання залікової контрольної роботи та РГР переводиться до залікової оцінки згідно з таблицею 1.

Таблиця 1.2 – Системи переведення балів до залікової оцінки

Бали, R	***ECTS оцінка***	***Залікова оцінка***
95-100	*A*	*відмінно*
85-94	*B*	*добре*
75-84	*C*	
65-74	*D*	*задовільно*
60-64	*E*	
Менше 60	*Fx*	*незадовільно*
РГР не зараховано	***F***	***не допущено***

1.3 Підготовка до лекцій

1.3.1 Загальні рекомендації

Лекція, як вид навчального заняття, не передбачає окремих процедур контролю підготовленості студентів. Проте це не означає, що студент має прийти на лекцію непідготовленим. Готуючись до кожної лекції з дисципліни «Проектування програмних доданків» студент повинен:

- ознайомитися з темою лекції та переліком питань, які будуть розглядатися на лекції;
- повторити попередньо освоєний матеріал, який потрібен для успішного засвоєння лекції;
- підготувати запитання з теми лекції, відповіді на які студент хотів би отримати;
- самостійно освоїти питання, які винесені на самостійне опрацювання за темою лекції.

Теми лекцій і перелік питань, що на них будуть розглядатися, наведені у цьому розділі. Для кожного питання наведені посилання на літературні джерела. Також наведений перелік матеріалів, знання яких є передумовою продуктивної роботи студента на лекції.

Запитання до теми лекції студент може готувати як письмово, так і в усній формі. Ймовірно, що в процесі читання лекції викладач дасть відповіді на питання, що зацікавили студента і частину питань буде, таким чином, знято. Ті запитання, що не знайшли відповіді у самій лекції, а також ті, відповіді на які не зрозумілі студенту, наполегливо рекомендується задати викладачеві у кінці лекції або на консультаціях.

Після прочитання кожної лекції студентам пропонується декілька питань на самостійне опрацювання за темою. Такі питання та перелік літературних джерел для їх освоєння наведені у цьому розділі.

1.3.2 Особливості програмування в C++/CLI (Лк 1)

Питання, які розглядаються на лекції:

- Специфіка C++/CLI: базові типи даних.

- Створення консольного додатку CLR.
- Виведення C++/CLI в командний рядок.
- Специфіка C++/CLI: форматування виведення.
- Відстежувані дескриптори.
- Відстежувані посилання.
- Загальні відомості про масиви.
- Сортування одномірних масивів.
- Пошук в одномірному масиві.
- Багатомірні масиви.
- Масив масивів.
- Об'єкти класу String.
- Об'єднання рядків.
- Зміна рядків.
- Порівняння рядків.
- Пошук рядків.

*Література: [*1, с. 63-107, 127-142, 237-268, 2, с. 47-56, 3, с. 175-183*].*

Питання, які виносяться на самостійне опрацювання:

- Клавіатурне введення в C++/CLI.
- Застосування оператора safe_cast.
- Збирання «сміття» в C++/CLI.

Література: [1, с. 127-145].

1.3.3 Функції в програмах C++/CLI та програмування класів (Лк 2)

Питання, які розглядаються на лекції:

- Основні відмінності функцій CLR.
- Функції зі змінною кількістю аргументів.
- Особливості класів в C++/CLI.
- Визначення типів класів значень.
- Визначення класів посилань.
- Визначення конструктора копіювання для класів посилань.
- Властивості класів.

*Література: [*1, с. 270-310, 352-364, 418-442, 541-551*].*

Питання, які виносяться на самостійне опрацювання:

- Функція класу ToString().
- Визначення скалярних властивостей.
- Тривіальні скалярні властивості.
- Визначення індексованих властивостей.
- Статичні властивості.
- Поля initonly.
- Статичні конструктори.

*Література: [*1, с. 418-444*].*

1.3.4 Основи роботи в інтегрованому середовищі розробки (Лк 3)

Питання, які розглядаються на лекції:

- Інтегроване середовище розробки.
- Використання IDE.
- Технологія Windows Forms.
- Створення нового проекту CLR.
- Додавання форми у проект CLR.
- Створення шаблону проекту CLR.
- Основні вікна IDE.
- Робота з додатками Windows Forms.
- Додавання головного меню.
- Створення контекстного меню.
- Додавання панелі інструментів.

*Література: [*1, с. 29-62, 925-942, 3, с. 223-254, 288-440*].*

Питання, які виносяться на самостійне опрацювання:

- Інтерфейс API Windows.
- Створення інтерактивних додатків.
- Стандартні простори імен бібліотеки .NET.
- Зміна властивостей форми.

*Література: [*1, с. 29-62*]*

1.3.5 Робота з текстовими і бінарними файлами, редагування графічних даних (Лк 4)

Питання, які розглядаються на лекції:

- Діалогові вікна відкриття та збереження файлу.
- Перехоплення помилок оператором try catch.
- Діалогове вікно повідомлень класу MessageBox.
- Простий текстовий редактор.
- Подія форми Closing.
- Зчитування та запис бінарних файлів з використанням потоку даних.
- Табличне введення даних.
- Інструмент для створення файлу XML.
- Побудова графіку за табличними даними з використанням елементу Chart.
- Використання елементу PictureBox для відображення растрового файлу з можливістю прокрутки.
- Друк графічного файлу.
- Побудова графіку методами класу Graphics.

*Література: [*1, с. 993-1007, 2, с. 77-177, 3, с. 282-287*].*

Питання, які виносяться на самостійне опрацювання:

- Створення простого RTF-редактору.
- Функція GetExtension().
- Малювання в формі графічних примітивів (фігур).
- Вибір кольору з використанням переліку.
- Малювання у формі покажчиком миші.

*Література: [*2, с. 77-109*]*

1.3.6 Робота з базами даних та використання функцій зовнішніх програм (Лк 5)

Питання, які розглядаються на лекції:

- Технологія ADO.NET.
- Створення бази даних MS Access.
- Запис структури таблиці в порожню базу даних MS Access.
- Додавання записів в таблицю бази даних MS Access.
- Зчитування всіх записів з таблиці бази даних за допомогою об’єктів класів Command, DataReader та елементу керування DataGridView.

- Зчитування даних з БД в сітку даних DataGridView з використанням об’єктів класів Command, Adapter та DataSet.
- Оновлення записів в таблиці бази даних MS Access.
- Перевірка правопису засобами MS Word.
- Виведення таблиці у MS Word.
- Використання фінансової функції MS Excel.
- Рішення системи рівнянь за допомогою функцій MS Excel.
- Побудова діаграм засобами MS Excel.

*Література: [*2, с. 190-251*].*

Питання, які виносяться на самостійне опрацювання:

- Видалення записів з таблиці бази даних з використанням SQL-запиту і об'єкта класу Command.

*Література: [*2, с. 229-241*]*

1.4 Підготовка до комп'ютерних практикумів

1.4.1 Загальні вимоги та рекомендації

Комп'ютерні практикуми проводяться у комп'ютерних класах із окремими групами студентів. У випадку великої кількості студентів у групі, вона може поділятися на дві підгрупи і заняття проводяться у двох класах одночасно. Метою комп'ютерних практикумів є вироблення та закріплення умінь та навичок програмування мовою `Visual C++ CLR`.

Студент зобов'язаний ретельно готуватися до кожного з комп'ютерних практикумів. Самостійна робота студента при підготовці до комп'ютерного практикуму з дисципліни «Прикладне програмне забезпечення - 3. Проектування програмних доданків» передбачає:

- освоєння теоретичного матеріалу з теми практикуму на основі лекцій та літературних джерел;
- відповіді на питання контролю підготовленості до практикуму;
- підготовку звіту з практикуму.

Перелік теоретичного матеріалу, який необхідно освоїти та питання для контролю підготовленості до кожного з практикумів наведені у цьому розділі. Звіт із комп'ютерного практикуму оформлюється на аркушах формату А4 і має містити:

- *титульний аркуш* із зазначенням номеру практикуму, його теми, групи і прізвища студента, який виконав роботу та прізвища викладача, який її перевірив (зразок титульного аркуша наведено в додатку А);
- *мету* та *завдання* практикуму;
- *короткі теоретичні відомості* для висвітлення питань, яким присвячена робота;
- *порядок виконання* завдань практикуму, з зазначенням виконуваних дій та операцій;
- *результати виконання* завдань у роздрукованому вигляді;
- *код програмного модуля*.

Перші три пункти цього переліку готуються студентом заздалегідь і пред'являються викладачу на початку практикуму. Це є необхідною умовою його успішного виконання. Результати практикуму друкуються, очевидно, після виконання завдань і перевірки їх викладачем. Результати роботи створеної

програми, а також код програмного модулю слід друкувати як частину протоколу на аркушах формату А4.

Виконання кожного з комп'ютерних практикумів проводиться за методикою, викладеною у відповідних методичних вказівках [7]. Порядок оцінювання комп'ютерних практикумів наведено в розділі 1.2.

1.4.2 Робота з масивами в консольному додатку CLR (КП 1)

Мета: вивчити прийоми створення нового консольного додатку CLR в Visual C++. Засвоїти методи виведення та введення даних в консольному додатку CLR. Вивчити засоби управління форматом виведення даних в консольних додатках CLR. Ознайомитись з одномірними та багатомірними масивами мови C++/CLI. Відпрацювати навички пошуку значень в масивах C++/CLI. Ознайомитись з прийомами сортування даних в масивах C++/CLI.

Завдання: створити новий консольний додаток CLR, додати в нього необхідний програмний код для створення двомірного масиву, введення необхідних значень та виведення масиву на екран після реалізації поставленої задачі згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Що таке середовище CLR? Як створити консольний додаток CLR?
2) Для чого потрібна функція `WriteLine()` мови C++/CLI? В якому просторі імен вона знаходиться?
3) Для чого потрібна функція `Write()` мови C++/CLI? В якому просторі імен вона знаходиться?
4) Як організувати форматоване виведення текстових рядків за допомогою функцій `Write()` та `WriteLine()`?
5) Для чого використовується специфікація формату у вигляді `{n,w:Axx}` та що вона означає?
6) Наведіть основні специфікатори форматування консольного виведення C++/CLI?
7) Для чого використовуються функції `Console::ReadLine()` та `Console::Read()`?
8) Для чого використовується функція `Console::ReadKey()`?
9) Для чого використовується функція `Parse()`?
10) В чому відмінність масивів C++/CLI від масівів «рідного» C++?
11) Як створити масив C++/CLI?
12) Для чого використовується цикл `for each` середовища CLR?

13) Для чого використовується функція `Array::Clear()` та як вона працює?
14) Як задати масив випадкових чисел?
15) Як організувати сортування одномірного масиву?
16) Як організувати сортування даних у двох пов'язаних масивах?
17) Для чого використовується функція `BinarySearch()` та як вона працює?
18) Як задаються багатомірні масиви в C++/CLI? Який максимально можливий ранг масиву?

1.4.3 Створення додатку Windows Forms (КП 2)

Мета: вивчити прийоми створення нового додатку `Windows Forms` в `Visual C++`. Засвоїти методи *компіляції*, *запуску* та *відлагоджування* проекту `Windows Forms`. Ознайомитись з класами `Form`, `GroupBox`, `Panel`, `Lable`, `TextBox`, `Button`, `RichTextBox`, їх властивостями та методами. Вивчити оператори перетворення типів при роботі з проектами `Windows Forms` в `Visual C++`.

Завдання: створити новий пустий проект `CLR`, додати в нього форму `Windows Forms`. Створити шаблон додатку Windows Forms. Додати необхідні елементи інтерфейсу у форму згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Що таке додаток `Windows Forms`?
2) Як створити новий проект `Windows Forms`?
3) Як додати нову форму у проект `Windows Forms`?
4) Як створити шаблон проекту?
5) Що таке `Конструктор` форми, як додавати нові елементи інтерфейсу у форму?
6) Для чого потрібне вікно `Редактора`?
7) Для чого потрібен `Оглядач рішень`?
8) Як змінити властивості об'єкту? Як додати реакцію на подію для об'єкту?
9) Як організувати перетворення типів при зчитуванні введених даних з об'єкту `TextBox`?
10) Як організувати перетворення типів при виведенні результатів розрахунку в об'єкт `RichTextBox`?
11) Як створити діалогове вікно з повідомленням?

1.4.4 Створення меню та панелей інструментів (КП 3)

Мета: вивчити прийоми створення меню та панелей інструментів в новому додатку Windows Forms в Visual C++. Засвоїти методи *додавання* та *видалення* компонентів меню та панелей іструментів у проекті Windows Forms, *зміну властивостей* об'єктів головного, контекстного меню та панелей інструментів, *призначення обробників подій* виклику пунктів меню чи команд панелі інструментів. Ознайомитись з елементами управління MenuStrip і ContextMenuStrip та їх об'єктами MenuItem, ComboBox, Separator та TextBox, їх призначенням та властивостями. Ознайомитись з елементом управління ToolStrip та його об'єктами Button, Label, SplitButton, DropDownButton, Separator, ComboBox, TextBox, ProgressBar, їх призначенням та властивостями.

Завдання: створити новий пустий проект CLR, додати в нього форму Windows Forms. Додати у форму елементи головного меню, контекстного меню та панелі інструментів. Додати необхідні елементи інтерфейсу та програмну реалізацію поставленої задачі згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Як додати об'єкт головного меню у форму? Які елементи може містити головне меню? В чому їх призначення?
2) Як додати об'єкт панелі інструментів у форму? Які елементи може містити панель інструментів? В чому їх призначення?
3) Перерахуйте основні властивості пунктів меню та їх призначення.
4) Перерахуйте основні властивості елементів панелі інструментів та їх призначення.
5) Елементами якого класу є пункти меню? Як додати обробник подій до пункту меню?
6) Чи може форма містити більше одного об'єкту головного меню? Якщо так, то для чого це потрібно?
7) Як створити контекстне меню? Як організувати появу різних контекстних меню для різних візуальних об'єктів форми?

1.4.5 Табличне введення даних (КП 4)

Мета: вивчити прийоми роботи з таблицями з використанням об'єкту DataGridView. Засвоїти методи створення, збереження та відкриття таблиць у форматі файлу XML. Ознайомитись з роботою об'єкту DataSet у проектах

Windows Forms у Visual C++. Навчитися створювати об'єкти DataTable для збереження даних у табличному вигляді. Ознайомитись з прийомами побудови графіків з використанням об'єкту Chart.

Завдання: створити новий пустий проект CLR, додати в нього форму Windows Forms. Додати необхідні об'єкти з Панелі елементів для створення головного та контекстного меню, а також панелі інструментів. Додати до форми об'єкт DataGridView для відображення таблиці результатів у формі. Додати об'єкт Chart для створення графіків на основі табличних даних. Створити об'єкти потрібних класів для реалізації поставленої задачі згідно отриманого варіанту завдання. Передбачити можливість збереження таблиці даних у обраний користувачем файл формату XML. Надати користувачу можливість відкривати набір даних з вже існуючого файлу формату XML. Організувати побудову графіків на основі стовпців створеної таблиці з використанням об'єкту Chart.

Питання для контролю підготовленості до практикуму:

1) Для чого використовується об'єкт DataGridView?
2) Для чого використовується об'єкт DataTable?
3) Для чого використовується об'єкт DataSet?
4) Як організувати виведення табличних даних у вікні додатка?
5) Як можна зберегти дані таблиці? В якому форматі вони зберігаються та як їх можна переглянути?
6) Для чого використовуються об'єкт Chart? Наведіть його основні можливості.
7) Наведіть типи діаграм, які можна побудувати за допомогою об'єкту Chart.
8) Яким чином зв'язати табличні дані з об'єктом Chart?
9) Як можна програмно змінювати тип діаграми об'єкту Chart?

1.4.6 Редагування графічних даних (КП 5)

Мета: вивчити прийоми роботи з графічними даними. Засвоїти методи створення, збереження та відкриття файлів у графічних форматах. Ознайомитись з роботою об'єкту класу Graphics для роботи з графічною інформацією у проектах Windows Forms у Visual C++. Навчитися створювати об'єкти класів Graphics та PictureBox для відображення графічних даних у вікні програми. Ознайомитись з прийомами побудови графіків з використанням об'єкту Graphics. Вивчити прийоми роботи з об'єктом створення багатосторінкових документів TabControl.

Завдання: створити новий пустий проект CLR, додати в нього форму Windows Forms. Додати необхідні об'єкти з Панелі елементів для створення головного, контекстного меню та панелі інструментів. Додати об'єкт TabControl у форму. Створити три вкладки в об'єкті TabControl. На першу вкладку додати об'єкти потрібних класів для введення необхідних вихідних даних згідно отриманого варіанту завдання. На другу вкладку додати об'єкт DataGridView для виведення результатів розрахунку у вигляді таблиці значень. На третю вкладку додати об'єкт PictureBox для відображення графічної залежності. В об'єкті PictureBox побудувати графік вказаної залежності засобами класу Graphics. Передбачити можливість збереження отриманої табличної залежності у обраний користувачем файл формату XML та відкриття вже існуючих файлів XML. Передбачити можливість збереження отриманої графічної залежності у обраний користувачем графічний файл, а також відкриття довільного графічного файлу у об'єкті PictureBox без зміни масштабу з можливістю прокрутки зображення.

Питання для контролю підготовленості до практикуму:

1) Для чого використовується метод OnPaint?
2) Опишіть метод DrawImage та його параметри.
3) Для чого використовується об'єкт класу Graphics?
4) Опишіть призначення та принцип використання методів DrawString, DrawLine та DrawEllipse класу Graphics.
5) Для чого використовується об'єкт класу PictureBox? Опишіть його властивості.
6) Опишіть призначення та принцип роботи об'єкту класу OpenFileDialog. Які властивості він має?
7) Опишіть призначення та принцип роботи об'єкту класу SaveFileDialog. Які властивості він має?
8) Опишіть призначення властивості Dock для деяких візуальних об'єктів. Перерахуйте можливі значення даної властивості та очікуваний результат.
9) Для чого використовується властивість Filter для об'єктів класів OpenFileDialog та SaveFileDialog? Яким чином присвоюються значення для цієї властивості?
10) Для чого використовуються об'єкти класу TabControl? Поясніть принцип їх роботи.
11) Для чого використовуються об'єкти класу TabPage? Поясніть принцип роботи з ними.

12) Для чого застосовується функція GetFileName() та які особливості її використання?
13) Для чого використовуються функції FromFile() та LoadFile().

1.4.7 Робота з базами даних (КП 6)

Мета: вивчити прийоми роботи з базами даних в середовищі CLR. Навчитись створювати бази даних формату *.mdb. Засвоїти прийоми редагування таблиць бази даних з використанням об'єкту DataGridView. Навчитись створювати прості запити на вибірку даних з таблиці.

Завдання: створити новий пустий проект CLR, додати в нього форму Windows Forms. Додати необхідні об'єкти з Панелі елементів для створення головного та контекстного меню, а також панелі інструментів. Додати до форми потрібні об'єкти класів Panel, Button, DataGridView для створення графічного інтерфейсу. Програмним шляхом створити базу даних формату MS Access *.mdb та таблицю в ній. Програмно створити структуру таблиці не менше ніж з 6 полів, вказавши поля та їх типи в новій таблиці. Розробити СКБД, яка дозволятиме переглядати та редагувати і додавати дані до БД з використанням об'єкту DataGridView. Наповнити таблицю БД інформацією, що міститиме не менше 30 записів. Передбачити поле для створення простих запитів на вибірку даних, в тому числі відбір записів за умовою та видалення записів за умовою.

Питання для контролю підготовленості до практикуму:

1) Що таке технологія ADO.NET?
2) Наведіть переваги технології ADO.NET?
3) Що таке постачальник даних .NET Framework? Які постачальники даних включає .NET Framework?
4) Як додати до проекту DLL-бібліотеки ADO?
5) Як програмно створити нову базу даних MS Access *.mdb?
6) Як записати структуру нової таблиці в базу даних MS Access за допомогою технології ADO.NET?
7) Як сформувати SQL-запит за допомогою екземпляру класу OleDbCommand?
8) Для чого використовується екземпляр класу OleDbConnection?
9) Для чого використовується метод ExecuteNonQuery()?
10) Як додати записи до таблиці бази даних?
11) Як перевірити виконання підключення до бази даних в режими проектування?

12) Як створити новий запит за допомогою Оглядача серверів?
13) Як вивести таблицю БД в об'єкт класу DataGridView за допомогою об'єктів класів Command та DataReader?
14) Як вивести таблицю БД в об'єкт класу DataGridView за допомогою об'єктів класів Command, Adapter та DataSet?
15) Як організувати редагування записів в таблиці бази даних?
16) Як видалити записи з таблиці бази даних за допомогою SQL-запиту?

1.4.8 Використання функцій зовнішніх програм (КП 7)

Мета: вивчити прийоми роботи з функціями зовнішніх програм. Ознайомитись з деякими бібліотеками зовнішніх програм, навчитися їх підключати до програмного модуля та використовувати функції зовнішніх бібліотек для більш ефективного вирішення поставлених задач.

Завдання: створити новий пустий проект CLR, додати в нього форму Windows Forms. Додати необхідні зовнішні бібліотеки. Перенести потрібні об'єкти з Панелі елементів для створення головного та контекстного меню і панелі інструментів. Додати до форми інші об'єкт, необхідні для створення інтерфейсу користувача для вирішення поставленої задачі. Створити об'єкти потрібних класів для реалізації поставленої задачі згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Яку бібліотеку потрібно підключити для використання функцій MS Word в програмі?
2) Яку бібліотеку потрібно підключити для використання функцій MS Excel в програмі?
3) Як підключити зовнішню бібліотеку до проекту?
4) Яка функція дозволяє перевірити орфографії засобами MS Word?
5) Як можна записати значення в комірки MS Excel?
6) Яка функція відповідає за вибір типу діаграму MS Excel?
7) Як приховати від користувача використання можливостей MS Excel в програмі?
8) Як організувати збереження у обраному користувачем файлі XML заданий у об'єкті DataSet набір даних?
9) Як додати в програму можливість використання бібліотеки функцій Microsoft VisualBasic?

1.5 Самостійна робота

1.5.1 Перелік тем для самостійного вивчення

Середовище .NET Framework:

- Загальномовне виконуюче середовище (CLR).
- Програмування, що керується подіями.
- Інтерфейс API Windows.
- Створення інтерактивних додатків.

*Література: [*1, с. 29-37, 801-835*].*

Інтегроване середовище розробки:

- Використання IDE.
- Основні вікна IDE Visual C++.
- Ініціалізація і обробка подій миші і клавіатури.

*Література: [*1, с. 37-62, 2, с. 57-76*].*

1.6 Розрахунково-графічна робота

1.6.1 Вимоги до виконання та оформлення роботи

Розрахунково-графічна робота виконується студентом самостійно в час, вільний від занять. Із усіх питань, які виникають у процесі виконання розрахунково-графічної роботи, студент може звернутися до викладача у час, відведений для проведення консультацій.

Завдання на розрахунково-графічну роботу з дисципліни «Прикладне програмне забезпечення - 3. Проектування програмних доданків» видається студентам після прослуховування переважної частини лекційного курсу, зазвичай на 6-7 тижні семестру. Видача завдань та закріплення варіантів розрахунково-графічної роботи обов'язково фіксується у листі реєстрації індивідуальних завдань.

Час виконання роботи складає один місяць, рахуючи з дати видачі завдання. Протягом всього цього терміну студенти можуть звертатися до викладача за консультаціями щодо неї. Після завершення терміну виконання, консультації з питань розрахунково-графічної роботи завершуються і студентам надається додатково 3-4 дні для здачі роботи. У разі, якщо студент не дотримав термінів здачі роботи, він отримує до свого семестрового рейтингу штрафні рейтингові бали (див. розділ 1.2).

Виконана робота здається викладачу в роздрукованому вигляді з додаванням електронної версії програмного модуля свого варіанту завдання. Роздрукований примірник розрахунково-графічної роботи має складатись з:

1) *Титульного аркушу*;
2) *Завдання* (загальної частини та індивідуального завдання);
3) *Теоретичних відомостей* (коротко описуються застосовані при виконанні теоретичні аспекти);
4) *Ходу виконання* (описується створення програмного модулю, описується алгоритм роботи з програмним модулем, наводиться приклад застосування програмного модулю та результати розрахунку);
5) *Коду програмного модулю* (наводиться повний код програмного модуля з усіма окремими файлами модулів `*.cpp` та файлами заголовку `*.h`).

До оголошення оцінок за розрахунково-графічну роботу студент повинен зберігати електронні варіанти проекту з вихідними файлами виконаного завдання.

1.6.2 Методичні рекомендації до виконання роботи

З метою закріплення отриманих знань та навичок, у рамках кредитного модулю передбачене виконання розрахунково-графічної роботи. Розрахунково-графічна робота виконується за темою: «Створення незалежного додатку `Windows Forms` з застосуванням власного класу». Для виконання роботи студентам видаються додому індивідуальні завдання. За результатами розрахунково-графічної роботи робиться висновок про достатнє (недостатнє) володіння студентом матеріалом дисципліни.

Мета: вивчити принципи створення власних *класів* та роботи з ними: *визначення класу*; створення *конструктору класу*; створення *інтерфейсу класу*; створення *реалізації класу*; *об'явлення функцій-членів класу*; виклик *функції-члену* класу; створення змінної типу власного класу; передача аргументів *функціям-членам* класу; створення програм з обчислювальною частиною, яка реалізована у власному класі та його *функціях-членах*.

Завдання: створити додаток `Windows Forms` з програмним інтерфейсом, який міститиме головне та контекстне меню, панель інструментів та інші елементи інтерфейсу для реалізації розрахунку з використанням математичного методу у вигляді класу користувача, згідно отриманого варіанту завдання.

Загальні вимоги.

1) Створити додаток `Windows Forms` з ім'ям виду *PrizvischeRGR*.
2) Програмний код модуля моє бути чітко структурований.
3) Імена об'єктів мають нести сенсові навантаження.
4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

1) Створити за допомогою *файлу заголовку* власний *клас даних*, який реалізуватиме розрахунок відповідним математичним методом (згідно варіанту).
2) Введення вхідних даних реалізувати в основному тілі програми.
3) Всі етапи обчислення реалізувати у власному класі.
4) У класі передбачити три *конструктори*:

a) конструктор для ініціалізації класу всіма необхідними даними (коефіцієнти рівняння, інтервал, точність);
b) для ініціалізації класу лише мінімально необхідними даними (наприклад, у реалізації класу передбачити ініціалізацію меж інтегрування та точності інтегрування значеннями за замовчуванням);
c) ініціалізацію класу конструктором за замовчуванням без параметрів (для запуску тестового варіанту розрахунку з вхідними даними за замовчуванням).

5) У класі створити:
   a) *інтерфейс класу* (відкриту для користувача частину);
   b) *реалізацію класу* (інкапсульовану всередині класу його частину);
   c) інтерфейсна частина класу та частина його реалізації повинні мати не менше однієї функції-члену кожна.
6) У програмі передбачити можливість вибору користувача:
   a) вводити всі вхідні дані (наприклад, коефіцієнти рівняння, межі інтегрування, точність інтегрування) для ініціалізації об’єкту класу за допомогою розширеного конструктора;
   b) вводити лише мінімально необхідні дані (наприклад, інтервал інтегрування та точність) для виклику конструктора класу, який передає ініціалізацію інших вхідних даних значеннями за замовчуванням у блоці реалізації класу;
   c) запуск тестового варіанту розрахунку з використанням *конструктора за замовчуванням* без введення вихідних даних в основному модулі програми.
7) Передбачити виведення результатів розрахунку на кожній ітерації, а також знайденого результату з заданою точністю на екран та, за бажанням користувача, у зовнішній файл за допомогою діалогового вікна `SaveFileDialog` та фільтру файлів у ньому.
8) Програмний модуль має містити об’єкт класу `Chart` або графічний об'єкт класу `Graphics` для побудови графічної залежності.
9) Побудована графічна залежність повинна зберігатися у зовнішній графічний файл з використання діалогового вікна `SaveFileDialog` та фільтру файлів у ньому.
10) В програмі має бути передбачена можливість відкриття збережених файлів результатів розрахунку та файлу графічної залежності з використанням діалогового вікна `OpenFileDialog` та фільтру файлів у ньому.
11) Інтерфейс програмного додатку повинен будуватись з використанням об’єкту `TabControl` та не менше трьох сторінок `TabPage`:

a) сторінка введення вихідних даних;
b) сторінка результатів розрахунку в якій формується звіт;
c) сторінка побудови графічної залежності.

12) Програмний додаток повинен містити головне меню, кілька контекстних меню (не менше двох), з прив'язкою до певної сторінки TabPage, та панель інструментів.

13) Елементи меню повинні містити:
a) назву пункту меню українською мовою;
b) піктограму для візуальної підказки призначення пункту меню;
c) сполучення гарячих клавіш для виклику події пункту меню.

14) Панель інструментів повинна містити набір команд для реалізації всіх основних можливостей програми. При цьому кожна кнопка панелі інструментів повинна мати:
a) піктограму для візуальної підказки призначення кнопки панелі інструментів;
b) спливаючу підказку українською мовою для пояснення призначення даної кнопки.

15) Візуальні елементи інтерфейсу програмного додатку мають бути структуровані у формі з використанням елементів Panel та інших контейнерів, а також закріплені у своїх елементах-контейнерах за допомогою властивостей Dock та Anchor таким чином, щоб при зміні розміру вікна інтерфейс програми не спотворювався.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Функції

2.1.1 Основні відмінності функцій CLR

В основному функції програм C++/CLI працюють точно так же, як і в програмах «рідної» C++. Звичайно, тут доводиться мати справу з дескрипторами і відстежуваними посиланнями замість звичайних покажчиків і посилань, що веде до деяких відмінностей. Оскільки існує кілька відмінностей, систематизуємо їх.

- Параметри функцій і повернене значення в програмі CLR можуть бути типами класів значень, відстежуванми дескрипторами, відстежуваними посиланнями і внутрішніми покажчиками.
- Коли параметром є масив, немає необхідності в додатковому параметрі, що передає його довжину, оскільки масиви C++/CLI внутрішньо зберігають інформацію про власну довжині у властивості Length.
- У програмах C++/CLI не можна застосовувати адресну арифметику з параметрами-масивами, як це робиться в програмах на «рідній» C++, тобто завжди потрібно використовувати індексацію масивів.
- Повернення дескриптора області, виділеної в розподіленій пам'яті CLR, - не проблема, тому що збирач «сміття» подбає про звільнення пам'яті, коли необхідність в ній відпаде.
- Механізм отримання змінного списку аргументів в C++/CLI відрізняється від механізму «рідної» C++.
- Доступ до аргументів командного рядка в функції main() програми C++/CLI також відрізняється від механізму «рідної» C++.

Останні дві відмінності розглянемо докладніше.

2.1.2 Функції зі змінною кількістю аргументів

Стандарт C++/CLI дозволяє передачу змінної кількості аргументів, дозволяючи вказати список параметрів як масив з попередніми трьома крапками. Приклад функції зі змінною кількістю параметрів наведено нижче.

```
int sum(...array<int>^ args)
{
      // Код sum
}
```

Функція sum() отримує будь-яку кількість аргументів типу int. Для обробки аргументів ви просто звертаєтеся до елементів масиву args. Оскільки це масив CLR, кількість його елементів записано у властивості Length, і у вас немає проблем з визначенням кількості аргументів в тілі функції. Цей механізм являє собою удосконалення подібного механізму «рідної» C++, оскільки є безпечним щодо типів. Тип аргументів, одержуваних цією функцією, чітко вказано як int. Розглянемо приклад, щоб побачити цей механізм CLR в дії.

```
// Func1.cpp: главный файл проекта.
// Передача у функцію змінної кількості аргументів

#include "stdafx.h"

using namespace System;

double sum(... array<double>^ args)
{
	double sum(0.0);
	for each(double arg in args)
		sum += arg;
	return sum;
}

int main(array<System::String ^> ^args)
{
	Console::WriteLine(sum(2.0, 4.0, 6.0, 8.0, 10.0, 12.0));
	Console::WriteLine(sum(1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8,
9.9));
	return 0;
}
```

Цей приклад виводить наступне.

```
42
49,5
```

Функція sum() отримує аргументи типу double. Три крапки попереду параметру повідомляють компілятору те, що можна очікувати будь-яку кількість аргументів і що ці аргументи містяться в масиві елементів типу double. Зрозуміло, без трикрапки функція при виклику очікувала б тільки один аргумент, який повинен був бути дескриптором масиву.

У порівнянні з «рідним» дане визначення функції sum() помітно простіше. Тут, у версії C++/CLI, зникають всі проблеми, пов'язані з типом і кількістю аргументів. Сума елементів накопичується в простому циклі for each, який перебирає всі елементи масиву.

Розглянемо використання функцій користувача в консольного додатку CLR на наступному прикладі.

Необхідно створити консольний додаток, який розраховує суму доданків певної залежності. Доданки на кожній ітерації та загальну суму необхідно вивести на екран.

Для збереження значення окремих доданків оголошуємо масив:

```
array<double>^ доданки;
```

Вихідні значення для розрахунків запитуємо у користувача та зберігаємо у відповідних змінних:

```
Console::WriteLine(L"Введіть значення аргументу A");
A = Double::Parse(Console::ReadLine());
Console::WriteLine(L"Введіть значення аргументу B");
B = Double::Parse(Console::ReadLine());
Console::WriteLine(L"Введіть кількість доданків n");
n = Int32::Parse(Console::ReadLine());
```

Після введення користувачем кількості доданків, створюємо масив для збереження їх розрахованих значень:

```
доданки = gcnew array<double>(n);
```

Значення окремого доданку будемо розраховувати у функції користувача:

```
double функція(double a, double b, int k)
{//Функція знаходить значення окремого доданка
      double чисельник, знаменник;
      чисельник = pow((a + k), 3) - 5 * k;
      знаменник = sqrt(3 * a) + k;
      return чисельник / знаменник;
}
```

Розрахунок доданків та їх збереження у масив доданки[ ] організовуємо у циклі, де викликаємо функцію користувача функція() та виводимо поточне значення доданку на екран у форматованому вигляді:

```
for (size_t i = 0; i < доданки->Length; i++)
{//Записуємо значення і-го доданка в масив
      доданки[i]= функція(A, B, i+1);
      Console::WriteLine(L"{0}-ий доданок = {1:F2}", i+1, доданки[i]);
}
```

Для знаходження значення суми всіх доданків використовуємо функцію користувача сума():

```
double сума(array<double>^ масивДоданків)
{//Функція сумує елементи переданого масиву
      double сум(0.0);
      for each(double доданок in масивДоданків)
            сум += доданок;
      return сум;
}
```

Виклик даної функції здійснюємо безпосередньо у функції WriteLine() для виведення суми доданків на екран:

```
Console::WriteLine(L"Сума доданків = {0:F2}",сума(доданки));
```

В якості аргумента функції передаємо масив з розрахованими значеннями доданків:

```
сума(доданки)
```

Повний код додатка наведений в нижче.

```
// Func1.cpp : main project file.

// Робота з функціями

#include "stdafx.h"
#include <math.h>
#include <windows.h>

using namespace System;

double сума(array<double>^ масивДоданків)
{//Функція сумує елементи переданого масиву
      double сум(0.0);
      for each(double доданок in масивДоданків)
            сум += доданок;
      return сум;
}

double функція(double a, double b, int k)
{//Функція знаходить значення окремого доданка
      double чисельник, знаменник;
      чисельник = pow((a + k), 3) - 5 * k;
      знаменник = sqrt(3 * a) + k;
      return чисельник / знаменник;
```

```
}

int main(array<System::String ^> ^args)
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	double A, B;
	int n, i = 1;
	//Оголошуємо масив для запису окремих доданків
	array<double>^ доданки;
	Console::WriteLine(L"Введіть значення аргументу A");
	A = Double::Parse(Console::ReadLine());
	Console::WriteLine(L"Введіть значення аргументу B");
	B = Double::Parse(Console::ReadLine());
	Console::WriteLine(L"Введіть кількість доданків n");
	n = Int32::Parse(Console::ReadLine());
	//Створюємо масив на n елементів
	доданки = gcnew array<double>(n);
	for (size_t i = 0; i < доданки->Length; i++)
	{//Записуємо значення i-го доданка в масив
		доданки[i]= функція(A, B, i+1);
		Console::WriteLine(L"{0}-ий доданок = {1:F2}", i+1,
доданки[i]);
	}
	Console::WriteLine(L"Сума доданків = {0:F2}",сума(доданки));
	return 0;
}
```

2.2 Класи

2.2.1 Об’єктно-орієнтоване програмування

Концепція об’єктно-орієнтованого програмування (ООП) включає в себе поняття об’єктів, класів, інкапсуляції, наслідування та поліморфізму.

Об’єкт – це деяка математично-програмно описана сутність, окремий елемент оточуючого світу, з яким ми зустрічаємося у повсякденному житті. Наприклад, ваша конкретна собака, ваш конкретний телевізор, ваш конкретний автомобіль – це все об’єкти. Реальні об’єкти дві характеристики: стан, який визначається набором властивостей об’єкту, та поведінка. Наприклад, пес має стан, що визначається певним набором його властивостей, таких як ім’я, колір шерсті, порода, характер та ін. А його поведінка визначається тим, що він в даний мамонта може гавкати, вихляти хвостом і т. ін. Поведінка об’єкту визначається набором функцій, що задані у об’єкті, які в ООП також звуть методами.

Клас – це деяке креслення, певний проект, з якого створюється об’єкт. В класі закладені властивості та поведінка майбутнього об’єкту, який отримують з класу, як з проекту. Наприклад, «`Автомобіль`» - це клас. Автомобіль «`Toyota Mirai`», який продається в автосалоні, - це об’єкт класу «`Автомобіль`», певний окремий його представник. «`Собака`» - це певний клас об’єктів з однаковими властивостями та можливостями, а «`Пес Сірко`» - це об’єкт або екземпляр цього класу, тобто певний окремий його представник, з яким ми безпосередньо можемо взаємодіяти.

Об'єктно-орієнтоване програмування - це спосіб програмування з орієнтацією на об'єкти. При такому способі створюються великі програмні утворення - класи, куди закладаються загальні властивості майбутніх об'єктів, які стануть виходити за певними правилами з цих класів, куди закладаються варіанти поведінки майбутніх об'єктів через створювані в класах параметричні програми (методи класів). Такий спосіб значно прискорює розробку програмного забезпечення і полегшує працю програміста. Згадаймо, що одним з перших засобів автоматизації праці програміста були так звані стандартні програми, які виконували дії що часто зустрічаються (наприклад, переклад десяткових чисел в двійкові, обчислення тригонометричних функцій і т. п.). Такі програми об'єднувалися в бібліотеки стандартних програм. Сьогодні на більш високому рівні ми маємо бібліотеки класів, які постачають нашим програмам необхідні їм об'єкти (малюнки, фотографії, засоби мультимедіа і т. п.), тим самим підвищуючи якість та ефективність сучасного програмування.

Програмно в класах задаються елементи, звані *властивостями*. Вони описують стан об'єкта і зберігаються в спеціальних елементах, які називаються *полями*. Поведінка ж об'єкта описується спеціальними функціями, які в класах

звуться *методами*. Коли властивості класу присвоюються якісь конкретні значення, то тим самим з класу створюється конкретний об'єкт. Об'єкти створюються спеціальним методом класу, званим *конструктором*.

Методи обробляють внутрішні стани об'єкту і забезпечують механізм взаємодії між об'єктами. Наприклад, візьмемо клас велосипедів. Властивостями, що характеризують стан об'єктів цього класу, будуть: поточна швидкість, поточний стан перемикача педалі (зміна швидкості), кількість шестерень і поточна шестерня, за рахунок якої швидкість змінюється. Методами, які змінюють стан велосипеда будуть: зміна шестерінки, перемикання педалі і зміна швидкості. Весь цей механізм зміни властивостей методами класу захований всередині самого класу. Такий принцип взаємодії елементів класу носить назву *інкапсуляції* даних. Це фундаментальний принцип об'єктно-орієнтованого програмування.

У чому ж фактична користь від механізму класів-об'єктів?

По-перше, забезпечується модульність програмування: вихідний код об'єкта написаний і підтримується незалежно від вихідних кодів інших об'єктів (тобто підвищується надійність всієї задачі, програма якої складається з ланцюжка таких незалежних модулів).

По-друге, механізм роботи такого модуля прихований всередині самого модуля і не відволікає програміста на з'ясування різних дрібних деталей алгоритму.

По-третє, є можливість багаторазового використання елемента (як і колись багаторазове використання бібліотеки стандартних програм).

По-четверте, забезпечується легка змінюваність елементів в загальній програмі (в додатку): якщо такий елемент виходить з ладу, його можна легко замінити аналогічним елементом, не руйнуючи всю задачу.

Отже, ООП базується на використанні класів. Використання класів - це основна відмінність мови C++ від мови C.

2.2.2 Класи в C++

Класи в мові C++ використовуються для визначення власних типів даних. Фундаментальними поняттями концепції класів є абстракція даних та інкапсуляція.

Абстракція даних (`data abstraction`) – програмний підхід, що заснований на розділенні інтерфейсу та реалізації. *Інтерфейс* (`interface`) класу складається з операцій, які користувач класу може виконати з його об’єктом. *Реалізація* (`implementation`) включає змінні-члени класу, тіла функцій, що складають інтерфейс, а також інші функції, котрі потрібні для визначення класу, проте не призначені для загального використання.

Інкапсуляція (encapsulation) забезпечує розділення інтерфейсу та реалізації класу. Інкапсульований клас приховує свою реалізацію від користувачів, які можуть використовувати інтерфейс, проте не мають доступу до реалізації класу.

Під *користувачами* класу в C++ маються на увазі розробники програмного коду, які використовують вже створений раніше іншими розробниками клас.

Клас, який використовує абстракцію даних та інкапсуляцію, називають *абстрактним типом даних* (abstract data type). Програмісти, які працюють із класом, не мають знати як внутрішньо працює цей тип. Вони можуть розглядати його як абстракцію.

Для забезпечення інкапсуляції в C++ використовують *специфікатори доступу* (access specifier).

- Члени класу, які визначені після *специфікатора public*, доступні для всіх частин програми. *Відкриті члени* (public member) визначають *інтерфейс класу*.
- Члени, які визначені після *специфікатору private*, є *закритими членами* (private member), вони доступні для функцій-членів класу, але не доступні для коду, який цей клас використовує. Розділи private інкапсулюють (приховують) реалізацію.

На найпростішому рівні *структура даних* (data structure) – це спосіб групування взаємопов'язаних даних та стратегії їх використання.

Визначення класу починається із *ключового слова struct*, яке супроводжується ім’ям класу та (можливо пустим) тілом класу. *Тіло класу* обмежується фігурними дужками і формує нову *область видимості*. Визначені у класі імена мають бути унікальними в межах класу, але поза класом вони можуть повторюватись.

```
struct MyStruct //Створення власної структури даних (класу)
{
	std::string name;
	unsigned index;
	float sum;
};
```

За фігурною дужкою, що закриває тіло класу, має бути крапка з комою. Крапка з комою необхідна, оскільки після тіла класу можливо визначити змінні.

У тілі класу визначені члени (member) класу. В нашому прикладі у класу є лише *змінні-члени* (data member). Змінні-члени класу визначають вміст об'єктів цього класу. Кожен об’єкт класу має власний екземпляр змінних-членів класу.

У змінних-членів класу можна визначити *внутрішньокласовий ініціалізатор* (in-class initializer). Він використовується для ініціалізації

змінних-членів при створенні об'єктів. Члени без ініціалізаторів ініціалізуються за замовчуванням. Внутрішньокласові ініціалізатори мають бути укладені у фігурні дужки або йти за знаком =. Неможна визначити внутрішньокласовий ініціалізатор у круглих дужках.

```
struct MyStruct //Створення власної структури даних (класу)
{
      std::string name; // ініціалізатор за замовчуванням
      unsigned index = 1; // внутрішньокласовий ініціалізатор
      float sum = 1.0; // внутрішньокласовий ініціалізатор
      vector<int> vect{ 1,2,3,4,5 };   //внутрішньокласовий ініціалізатор
                                       // із використанням фігурних дужок
};
```

Мова C++ використовує для визначення власних структур окрім ключового слова `struct` також ключове слово `class`.

Для того, щоб скористатися створеною структурою даних, необхідно визначити об'єкт типу створеної структури. *Визначення об'єкту* типу класу в найпростішому випадку може бути аналогічне визначенню об'єктів інших типів.

```
string word; //визначення змінної вбудованого типу string
MyStruct my_str; //визначення змінної власного типу MyStruct
```

Після визначення змінної типу класу стає можливим звернення до членів даного класу. Звернення до членів класу використовується для отримання значень змінних-членів, зміни (присвоєння нового значення) значень змінних-членів, виконання певних операцій, які передбачені даним класом. Звернення до членів класу виконується за допомогою *оператору звернення до члену* (`.`).

```
int i = my_str.index; //значення i=1
my_str.sum = 5.5;
vector<int> vec1;
vec1 = my_str.vect; //vec1={ 1,2,3,4,5 }
```

Визначення класу у загальному вигляді з використанням ключового слова `class` виглядає наступним чином:

```
class MyClass //Ім'я класу
{
public:
      //Інтерфейс класу
private:
      //Реалізація класу
};
```

При визначені класу можна використовувати ключові слова `struct` або `class`. Відмінність у визначенні класу за допомогою цих ключових слів полягає у заданому за замовчуванням *рівні доступу*. Якщо використовується *ключове слово* `struct`, то члени, які визначені до першого специфікатора доступу, будуть *відкритими* (`public`), тобто входитимуть до інтерфейсу класу. Якщо використовується *ключове слово* `class`, то члени, які визначені до першого специфікатора доступу, будуть *закритими* (`private`), тобто входитимуть до реалізації класу.

Інкапсуляція надає дві важливі переваги.

- Код користувача не може за необачністю пошкодити інкапсульований об'єкт.
- Реалізація інкапсульованого класу може з часом змінитися і це не вимагатиме змін у коді на рівні користувача.

2.2.3 Змінні-члени та функції-члени класу

Змінні, які входять у визначення класу, називають *змінними-членами* (data member).

Члени класу, які є функціями, називають *функціями-членами* (member function).

Функції-члени визначають та об'являють як звичайні функції. Функції-члени *мають бути* об'явлені в класі, проте визначені вони *можуть* бути безпосередньо в класі або поза тілом класу. Функції, які не є членами класу, але є складовими інтерфейсу, об'являються та визначаються поза класом, але в тому ж самому файлі що і клас.

Лістинг 2.1

```
class MyClass
{
public: //Інтерфейс класу
     void Method(); //Об'явлення функції-члену класу без параметрів
private: //Реалізація класу
     float k1 = 1; //Ініціалізація даних-членів класу
     float k2 = 1;
     float k3 = 1;
     float a=-10;
     float b=10;
     double MyFunc(const double&); //Об'явлення функції-члену класу
                                   //з параметром - константним посиланням
     MyClass& MyFunc2(const MyClass&) //Об'явлення функції-члену класу
                 //з параметром - константним посиланням на тип класу
}; //Завершення тіла класу
```

```
void MyClass::Method() //Визначення функції-члену класу без параметрів
{                              //поза тілом класу
      double za, zb;
      za = MyFunc(a);
      zb = MyFunc(b);
}

double MyClass::MyFunc(const double &x) //Визначення функції-члену класу
{                  //з параметром – константним посиланням поза тілом класу
      double z;
      z = k1*exp(k2*x) + k3*x;
      return z;
}
```

Ім'я функції-члена, яка об'явлена в тілі класу, але визначена поза тілом класу, повинне включати ім'я класу, якому вона належить:

```
void MyClass::Method() //Визначення функції-члену поза тілом класу
```

Ім'я функції `MyClass::Method()` використовує оператор області видимості, щоб вказати, що дана функція об'явлена в межах класу `MyClass`.

Кожен клас визначає власну область видимості. Поза *областю видимості класу* (`class scope`) до звичайних даних і функцій його члени можуть звертатись лише через об'єкт, посилання або покажчик, використовуючи *оператор доступу до члену* (`.`). Для доступу до членів типу з класу використовується *оператор області видимості* (`::`). У будь-якому випадку наступне за оператором ім'я має бути членом відповідного класу.

У функції-члені можна безпосередньо звернутися до членів об'єкту, з якого вона була викликана. Для цього використовується покажчик `this`. *Параметр `this`* визначається неявно та автоматично і його можна використовувати в тілі функції-члену.

```
MyClass& MyClass::MyFunc2(const MyClass &rhs) //Визначення функції-члену
{          //класу з параметром – константним посиланням поза тілом класу
      a += rhs.a;
      return *this; //повертає об'єкт, для якого була викликана функція
}
```

Автори класів інколи визначають *допоміжні функції*, які використовуються як частина інтерфейсу класу, але при цьому членами класу вони не є. Такі функції об'являються (але не визначаються) в тому ж заголовку, що і сам клас після визначення класу. Таким чином, щоб використовувати будь-яку частину інтерфейсу класу, користувачу достатньо підключити лише один файл заголовку.

2.2.4 Конструктор класу

Кожен клас визначає, як можуть бути ініціалізовані об’єкти його типу. Клас контролює ініціалізацію об’єкту за рахунок визначення однієї чи декількох спеціальних функцій-членів, відомих як *конструктори* (constructor). Задача конструктора – ініціалізувати змінні-члени об’єкту класу. Конструктор виконується кожен раз, коли створюється об’єкт класу.

Конструктор класу теж член класу, але специфічний — це метод з таким же ім'ям, що й клас. Такі методи не мають право повертати якісь значення: якщо ви написали в конструкторі оператор `return`, компілятор видасть помилку. Конструктор виконує різні завдання і невидимий для вас як програміста. Навіть якщо ви самі не писали його текст, конструктор за замовчуванням завжди присутній у класі, бо основне його завдання — створювати в пам'яті примірник класу (тобто як би за проектом (класу) побудувати будинок (екземпляр)). У цьому випадку конструктор береться із загального прабатька класів — класу `Object`, в якому він є.

Друге завдання конструктора — надавати всім членам-даним класу початкові значення. Якщо ви самі не побудували конструктор, який стане ініціалізувати члени-дані вашого класу вашими ж значеннями, то цим даним невидимий для вас конструктор (конструктор за замовчуванням) надасть свої, прийняті в системі для даного типу величин значення (наприклад, дані типу `int` отримають нульові значення тощо).

Ім’я конструктора *співпадає* з іменем класу. На відміну від інших функцій, у конструкторів *немає типу* повернутого значення. Як і інші функції, конструктори мають список параметрів (можливо пустий) і тіло (можливо пусте). У класу може бути *декілька* конструкторів. Подібно будь-якій іншій перевантаженій функції, конструктори мають *відрізнятися* один від одного *кількістю* або *типами* своїх параметрів. Конструктори, на відміну від інших функцій, не можуть бути об’явлені константами.

Якщо в класі не визначено жодного конструктору, класи самі контролюють ініціалізацію відкритих змінних-членів значеннями за замовчуванням, визначаючи спеціальний конструктор, що зветься *стандартним конструктором* (`default constructor`). Стандартним вважається конструктор, який не отримує ніяких аргументів. У такому випадку змінні-члени ініціалюзуються внутрішньокласовими ініціалізаторами чи значеннями за замовчуванням.

Якщо клас не визначає конструктори явно, компілятор сам визначить стандартний конструктор неявно. Створений компілятором конструктор зветься *синтезованим стандартним конструктором* (`synthesized default constructor`).

Класи, члени яких мають *вбудований* чи *складений тип* (відповідно не мають ініціалізатора за замовчуванням), можуть розраховувати на синтезований стандартний конструктор, *тільки якщо у всіх* таких членів є внутрішньокласові ініціалізатори.

```
struct MyStruct
{
      int *pi = 0;     //Внутрішньокласовий ініціалізатор змінної
                       //складеного типу
      float y = 1;     //Внутрішньокласовий ініціалізатор змінної
                       //вбудованого типу
      double a;        //Визначення змінної-члену вбудованого типу
                       //без ініціалізатора
}; //Завершення тіла класу
```

Якщо у класу визначений хоча б один конструктор, то компілятор не буде створювати автоматично стандартний конструктор. У такому разі стандартний конструктор потрібно обов'язково визначити в класі самостійно.

Якщо в класі є інші конструктори, то можна попросити компілятор створити *стандартний конструктор* автоматично. Для цього після списку параметрів вказується частина = `default`.

```
MyClass() = default;
```

Оскільки цей конструктор не має параметрів – він є стандартним конструктором. Застосування цього конструктора можливе лише у випадку, коли для змінних-членів вбудованих та складених типів є *внутрішньокласові ініціалізатори*.

При створенні конструктору класу після переліку параметрів ставиться двокрапка, за якою розміщується *перелік ініціалізації конструктора* (`constructor initializer list`), який визначає вихідні значення для однієї чи декількох змінних-членів створюваного об'єкту. Завершується конструктор фігурними дужками, які містять тіло конструктора. *Ініціалізатор конструктору* – це перелік імен змінних-членів класу, кожне з яких супроводжується вихідним значенням у круглих (або фігурних) дужках. Якщо ініціалізацій кілька, вони відділяються комами.

```
MyClass() = default; //Стандартний конструктор
MyClass(const float &a1, const float &b1) : a(a1), b(b1) {};
//Конструктор класу
MyClass(const float &kof1, const float &kof2, const int &n, const float
&a1, const float &b1) : k1(kof1), k2(kof2), k3(kof2*n), a(a1), b(b1) {};
//Конструктор класу
```

У даному прикладі першим іде стандартний конструктор, який для ініціалізації змінних членів при створенні об’єкту класу використовує внутрішньокласові ініціалізатори (якщо вони є), в іншому разі використовує ініціалізацію змінних-членів значеннями за замовчуванням (окрім вбудованих та складених типів).

Другим іде конструктор класу, який ініціалізує змінні-члени a та b параметрами a1 та b1 відповідно. Змінні члени k1, k2 та k3 будуть ініціалізовані внутрішньокласовими ініціалізаторами.

Третій конструктор ініціалізує всі змінні-члени створюваного об’єкту класу переліком ініціалізації конструктора, який розміщений між двокрапкою та фігурними дужками. При цьому змінна-член k3 ініціалізується добутком параметрів kof2 та n.

Зазвичай для конструктора краще використовувати внутрішньокласовий ініціалізатор, якщо він є і присвоює члену класу вірні значення. Якщо компілятор не підтримує внутрішньокласову ініціалізацію (введена в стандарті C++11),кожен конструктор повинен явно ініціалізувати кожен член вбудованого типу.

У наведених вище конструкторів тіла пусті. Єдине їх призначення – присвоїти значення змінним-членам. Якщо нічого іншого робити не потрібно, то тіло функції залишається пустим.

Як і інші функції-члени, конструктори можуть визначатися як у тілі класу, так і за його межами. У тілі класу визначають функції, які складаються не більше як із одного-двох операторів, щоб не ускладнювати розуміння класу. Функції-члени, в тому числі конструктори, які мають більший обсяг коду у своєму тілі, об’являються в тілі класу, а визначаються одразу після нього в тому ж самому файлі заголовку. Перед іменем конструктора, який визначений за межами класу, як і перед іменем інших функцій-членів, вказується ім’я класу та оператор області видимості.

```
MyClass::MyClass()
{
	//Тіло конструктору
}
```

Ім’я класу перед іменем функції вказується для того, щоб зазначити що дана функція відноситься до вказаного класу. Оскільки ім’я функції співпадає з іменем класу, значить ця функція є конструктором зазначеного класу.

Деструктор класу це функція, зворотна суті функції конструктора. Вона покликана зруйнувати створений конструктором примірник класу і звільнити від нього пам'ять. Ім'я деструктора збігається з ім'ям класу, але перед ім'ям вказується знак «тильда» (~). Для попереднього прикладу деструктор буде мати вигляд: ~MyClass(). Деструктор у класу повинен бути один.

2.2.5 Дружні функції

Клас може дозволити іншому класу чи функції отримати доступ до своїх закритих членів, встановивши для них *дружні відносини* (`friend`). Клас об'являє функцію дружньою, включивши її об'явлення з ключовим словом `friend`.

Лістинг 2.2

```
class MyClass
{
friend float add(const float&);
friend std::ostream &print(std::ostream&, const MyClass&);
public: //Інтерфейс класу
      MyClass() = default; //Стандартний конструктор
      MyClass(const float &a1, const float &b1) : a(a1), b(b1) {};
      //Конструктор класу
      MyClass(const float &kof1, const float &kof2, const int &n, const
float &a1, const float &b1) : k1(kof1), k2(kof2), k3(kof2*n), a(a1),
b(b1) {}; //Конструктор класу
      void Method(); //Об'явлення функції-члену класу без параметрів
private: //Реалізація класу
      float k1 = 1; //Ініціалізація даних-членів класу
      float k2 = 1;
      float k3 = 1;
      float a=-10;
      float b=10;
      double MyFunc(const double&); //Об'явлення функції-члену класу
                                    //з параметром - константним посиланням
      MyClass& MyFunc2(const MyClass&) //Об'явлення функції-члену класу
                  //з параметром - константним посиланням на тип класу
}; //Завершення тіла класу
//Об'явлення частин, що не є складовими інтерфейсу класу
float add(const float&);
std::ostream &print(std::ostream&, const MyClass&);
```

Об'явлення друзів може розташовуватись лише *у визначенні* класу, використовуватись у класі вони можуть будь-де. Друзі не є членами класу і не підкоряються специфікатору доступу розділу, в якому вони об'явлені. Об'явлення друзів бажано групувати на початку чи в кінці визначення класу.

Об'явлення дружніх відносин встановлює лише права доступу. Це не об'явлення функції. *Для можливості виклику* дружньої функції користувачами

класу її слід також *об'явити*. Для доступу користувачів до дружніх функцій їх зазвичай об'являють поза класом, у тому ж заголовку, що і сам клас.

2.2.6 Визначення класів у заголовках

Класи зазвичай визначають у *файлах заголовку* (header). Файли заголовку мають розширення `*.h`. Як правило, класи зберігаються в заголовках, ім'я яких співпадає з іменем класу. Наприклад, бібліотечний тип `string` визначений у заголовку `string`. За аналогією наш клас `MyStruct` має бути визначений у заголовку `MyStruct.h`.

Заголовки підключаються до програмного файлу за допомогою директиви препроцесора `#include`. Коли препроцесор зустрічає директиву `#include`, він заміняє її вмістом файлу заголовку, який підключений за допомогою цієї директиви. З огляду на це, заголовки також не повинні містити об'явлення `using`, щоб запобігти включенню в програму не передбачених бібліотечних імен.

Заголовки містять сутності, які можуть бути визначені у будь-якому файлі лише раз. Для запобігання кількаразового включення вмісту одного й того самого заголовку, програми C++ використовують препроцесор для *захисту заголовку* (header guard). Захист заголовку опирається на змінні препроцесору. Змінні препроцесору можуть знаходитись у двох станах: визначена чи не визначена. Директива `#define` отримує ім'я і визначає його як змінну препроцесору. Існують дві директиви, що дозволяють перевірити чи була визначена змінна препроцесора. Директива `#ifdef` істинна, якщо змінна була *визначена*, а директива `#ifndef` істинна, якщо змінна *не була визначена*. При істинності перевірки виконується все, що розташоване після директиви `#ifdef` або `#ifndef` і до наступної директиви `#endif`.

Ці засоби можна використовувати для боротьби з багаторазовим включенням вмісту файлів заголовку наступним чином:

Лістинг 2.3

```
#ifndef MY_STRUCT_H
#define MY_STRUCT_H
#include <string>
struct MyStruct //Створення власної структури даних (класу)
{
```

```
        std::string name;
        unsigned index = 1;
        float sum = 0.0;
};
#endif
```

При першому включенні заголовку MyStruct.h директива #ifndef істинна, і препроцесор обробляє рядки після неї до директиви #endif. У результаті змінна MY_STRUCT_H буде визначена, а вміст заголовку MyStruct.h скопійовано до програми. Якщо надалі включити заголовок MyStruct.h в той же самий файл, то директива #ifndef виявиться брехнею і рядки між нею та директивою #endif будуть проігноровані.

Змінні препроцесора мають бути унікальними в усій програмі. Для цього в ім'я змінної препроцесора включається ім'я класу та ім'я змінної препроцесора задається лише великими літерами. Наприклад змінна препроцесора MY_STRUCT_H для включення визначення класу MyStruct із заголовку MyStruct.h.

Кожен заголовок повинен містити захист від повторного включення до програми його вмісту.

2.3 Класи середовища CLR

2.3.1 Загальні відомості про класи CLR

Клас - це звичайний тип. Якщо ви програміст, то завжди маєте справу з типами і екземплярами, навіть якщо і не використовуєте цю термінологію. Наприклад, ви створюєте різні змінні типу `int`. Можна сказати, що ви фактично створюєте різні екземпляри змінних цього типу. Класи зазвичай більш складні, ніж прості типи даних, але вони працюють тим же способом. Створюючи змінні типу заданого класу, ви створюєте *екземпляри* цього *класу*, а призначаючи різні значення екземплярів того ж типу (як і змінним типу `int`), ви можете виконувати різні завдання. Тому наша мета навчитися створювати та використовувати класами для створення програмного забезпечення.

Клас - це набір пов'язаної інформації, яка містить у собі і дані, і функції (програми для роботи з даними). Ці функції в класах називаються *методами*.

Клас - це конструкція, яка параметрично визначає деяку категорію об'єктів. Наприклад, може бути клас комп'ютерів, який об'єднує в собі комп'ютери різних марок, різних можливостей. Можливий клас столів: столи письмові, обідні і т. п. Клас столів може ділитися на підкласи: столи письмові, які, в свою чергу, можуть ділитися на столи письмові дубові і столи письмові деревоволокнисті і т. д. Ми бачимо, що класи можуть належати якійсь ієрархії класів. У кожному класі визначено характеристики тих об'єктів, які утворюють цей клас.

У класі також задаються програми, звані *методами*, які обробляють характеристики об'єктів, що належать даному класу. Поведінка об'єкта в реальному світі визначається його характеристиками. Змінюючи значення характеристик, ми отримуємо різну поведінку об'єктів. Коли ми створюємо екземпляр класу і визначаємо значення його конкретних характеристик, то отримуємо конкретний *об'єкт*.

Тобто ми маємо певний клас з набором *властивостей* та *методів*. Для роботи з даним класом потрібно створити окремого представника даного класу – *примірник* (*екземпляр) класу*. Надалі, для виконання потрібних нам завдань, треба присвоїти певні значення *властивостям* цього *екземпляру класу*, а також задіяти *методи* класу. В результаті отримуємо наш певний екземпляр класу, призначений для виконання конкретних завдань – *об'єкт класу*.

Наприклад, є клас `Стіл`. В нього є *дочірній* – породжений від нього клас `Стіл письмовий`, який *інкапсулював* в собі всі *властивості* та *методи* класу `Стіл` (наприклад, наявність столешні, ніжок, можливість сидіти за ним з

використанням об’єктів класу Стілець та ін.) та додав деякі нові властивості та методи, притаманні саме письмовим столам (розмір, форма, наявність шухляд, форма ніжок, висота столу та ін.).

При цьому методи класу Стіл мають властивість *поліморфізму*. Метод Сидіти за столом, що *інкапсульовано* у класі Стіл по різному працює у *породжених* від цього класу *дочірніх класах*. У класі Стіл кухонний метод Сидіти за столом передбачає прийняття їжі та можливість використання при цьому об’єктів класу Табурет кухонний чи Стілець кухонний. У дочірньому класі Стіл письмовий, який *успадкував* всі властивості та методи батьківського класу Стіл, успадкований метод Сидіти за столом передбачає виконання інших дій, ніж у класі Стіл кухонний, наприклад сидіти на об’єктах класів Стілець м’який чи Крісло комп’ютерне та писати на папері чи працювати за ноутбуком. *Інкапсуляція*, *спадковість* та *поліморфізм* – це основні поняття об’єктно-орієнтованого програмування (ООП).

Якщо нам потрібен письмовий стіл, ми обираємо певний проект письмового стола в меблевій майстерні – *екземпляр класу* Стіл письмовий. В меблевій майстерні нам виготовляють особисто для нас письмовий стіл на замовлення і ми отримуємо свій певний *об’єкт класу* Стіл письмовий, в якому потім використовуємо його властивості та методи у своїх цілях – розміщуємо на ньому об’єкти класів Лампа настільна, Ноутбук, кладемо папери та інші речі у шухляди, працюємо за ним сидячі на об’єкті класу Стілець м’який чи Крісло комп’ютерне.

У складі класу існує спеціальний метод (тобто програма-функція чи функція-член), який формує екземпляр класу. Цей метод носить назву *конструктора*. На противагу конструктору, існує програма *деструктор*, яка знищує екземпляр класу в пам'яті, щоб звільнити пам'ять, яка може використовуватися для інших програмних цілей. А якщо згадати, що пам'ять - величина не безмежна, то стає зрозумілою і роль деструктора.

У термінології ООП дані називаються *даними-членами* (змінними-членами), а програми, які їх обробляють (ці програми побудовані у вигляді функцій), *функціями - членами* (або методами).

В середовищі CLR (Common Language Runtime) Visual Studio введено поняття *компонентів* - спеціальних класів, в яких об'єкти визначаються такими характеристиками, як *властивості*, *події* і *методи*. Причому, на відміну від роботи зі звичайними класами, при роботі в середовищі CLR можливо маніпулювати виглядом і функціональною поведінкою компонентів і *на стадії проектування* (design) додатку, і в момент його *виконання* (runtime).

Наприклад, в CLR існує компонент «Форма» (клас Form) і компонент «Кнопка» (клас Button), у яких є свої *властивості* (properties), *події* (events) і *методи* (methods) – *обробники подій* (event handler), які є реакцією на події. У класі Button є властивість (property) Text, що містить напис на кнопці, подія (event) OnClick, яка викликається при щиглику лівої клавіші миші на кнопці, та реакція на подію (event handler) – метод Click, в якому записується програмний код для виконання необхідних дій при виникнення події OnClick. При цьому у кожного класу є свій певний набір подій, але не кожна подія об'єкту класу буде мати реакцію на подію – запрограмований метод. Так у класі кнопки Button є подія OnDoubleClick – подвійний щиглик миші на кнопці, але якщо для неї ми не напишемо метод DoubleClick – обробник даної події, то після подвійного щиглика миші по об'єкту button1 класу Button нічого не відбудеться, бо немає *обробника події*.

Якщо при проектуванні програми в форму помістити дві кнопки, то за допомогою визначення двох різних значень властивостям кнопок Text (назва кнопки) і Visible (значення false і true визначають видимість кнопки при виконанні додатка), ви отримуєте два примірника (екземпляри) класу, які ведуть себе по-різному: перша кнопка при виконанні програми буде невидима в формі, а друга залишиться видимою. За допомогою події компонент повідомляє користувачеві, що на нього виконано певний вплив (наприклад, для компонента «Кнопка» подією може бути натискання кнопки - клацання кнопкою миші), а методи служать для обробки реакції компонента на події.

Розглянемо структуру базового класу, з якого можуть створюватись класи нащадки. Об'явлення класу наведене в лістингу нижче.

Лістинг 2.4

```
class MyClass //Ім'я класу
{
public: //Ім'я секції
      //Данні і методи з цієї секції будуть доступні методам всіх класів
      MyClass(); //Конструктор класу
      ~MyClass(); //Деструктор класу
      <Загальнодоступні дані>
      <Загальнодоступні конструктори>
      <Загальнодоступні методи>
private: //Ім'я секції
      //Данні і методи з цієї секції будуть доступні
      //лише методам цього класу
      <Закриті дані>
      <Закриті конструктори>
      <Закриті методи>
```

```
protected: //Ім'я секції
      //Данні і методи з цієї секції будуть доступні методам цього класу
      //та похідним від нього, тобто нащадкам
      <Захищені дані>
      <Захищені конструктори>
      <Захищені методи>
};

MyClass::MyClass() //Конструктор класу
{
}

MyClass::~MyClass() //Деструктор класу
{
}
```

Атрибути `public`, `private` та `protected` називають *атрибутами доступу* до членів класу.

Дані та методи, які розміщені в секції `public` при визначенні класу будуть *загальнодоступними*, тобто до них можуть отримати доступ дані та методи як самого класу, так і з-за меж даного класу.

Методи і дані, що розміщені у секції класу `private` є *закритими*, тобто доступ до них можливий лише з цього класу для його даних та методів.

Все, що визначене в секції `protected` є *захищеним*, тобто до нього можуть отримати доступ всі дані та методи цього класу, а також всіх класів, які були *породжені* від нього. З інших класів та з-за меж даного класу вміст секції `protected` буде недоступний.

Поліморфізм – один з основних принципів створення класів. При поліморфізмі (дослівно: багатоформіє) споріднені об'єкти (тобто ті, що походять від спільного батька) можуть вести себе по різному в залежності від ситуації, яка виникне в момент виконання програми. Для досягнення поліморфізму потрібно мати можливість один і той самий метод з класу батька перевизначити в класі-нащадку.

Наприклад, всі класи мають загального прабатька - клас `Object`. В цьому класі визначено метод `Draw` (малювати фігуру). Класи, які малюють різні фігури і породжені від `Object`, - споріднені класи. Кожен з них визначає малювання своєї фігури методом `Draw`, успадкованим від `Object`: точки, лінії, прямокутника, кола і т. д. При цьому всі фігури різні, хоча метод загальний. Але цей метод `Draw` в кожному з класів-нащадків перевизначений, тобто в кожному класі-нащадку йому призначена інша функціональність.

Поліморфізм досягається за рахунок того, що методам з класу-батька дозволено виконуватися в класі-нащадку, де залишають тільки його ім'я, але при цьому дають йому необхідну для даного класу функціональність. Такі методи повинні оголошуватися в обох класах з атрибутом *virtual*, записуваним перед атрибутом «*тип даних, який повертається*». Якщо функція має атрибут virtual, то вона може бути перевизначена в класі-нащадку навіть якщо кількість і тип її аргументів такі ж, що і у функції базового класу. Перевизначена функція скасовує функцію базового класу.

Крім атрибута virtual, у методів існує атрибут *friend*. Методи з таким атрибутом, розташованим (як і атрибут virtual) в оголошенні методу перед зазначенням типу даних, які повертаються, і називаються *дружніми*. Метод, оголошений з атрибутом friend, має повний доступ до членів класу, які розташовані в секціях private і protected, навіть якщо цей метод не є членом цього класу. Це справедливо і для класів: зовнішній клас (тобто його методи) має повний доступ до класу, який оголошує цей зовнішній клас дружнім.

У всіх інших аспектах дружній метод - це звичайний метод. Подібні методи з зовнішніх класів, маючи доступ до секцій private і protected, можуть вирішувати завдання, реалізація яких за допомогою методів-членів даного класу ускладнена або навіть неможлива.

Наведемо приклади програм, в яких створені та використовуються найпростіші класи.

Лістинг 2.5

```
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>

class A
{
protected: //до цих даних мають доступ тільки методи даного класу
           // і похідних
     int x;
     int y;
public:
     int a;
     int b;
     int f1(int x, int y) //метод класу
     {
          return(x - y);
     }
};
```

```
class B : A // це оголошується клас, похідний від A
{           // при цьому успадковуються члени класу A
public:
      int f2(int x) //метод класу
      {
            A::x = 20; /* тут можуть використовуватися члени-дані базового
                  Класу з секції protected. Так як ім'я аргументу методу
                  f2() збігається  з ім'ям поля x з класу A,
                 успадкованого класом B, то щоб розрізняти яка змінна до
                 якого класу належить, треба було уточнити за допомогою
                  специфікатор ::. Показано, що в тілі методу x береться
                  той, що успадкований від A, і власний аргумент x
                  самого методу f2(): */
            return(x + A::x);
      }
};
```

Зверніть увагу, що коли ви створюєте змінну типу класу, тобто примірник (екземпляр) класу (наприклад, `A min;`, де `min` — примірник класу), то для цього примірника ніякої пам'яті не виділяєте, а компілятор все-таки розміщує цей примірник десь у пам'яті. Чому?

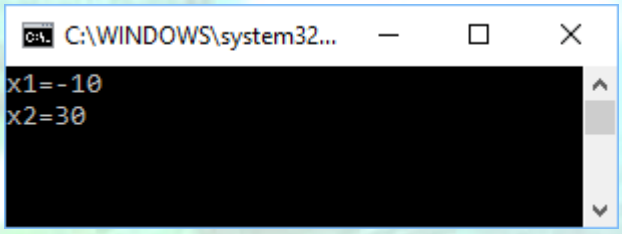


Рисунок 2.1 – Результат роботи програми з лістингу 6.5 та 6.6

Середовище виконання додатків в C/C++ виділяє два види пам'яті: *статичну* і *динамічну*. Остання носить назву "*купа*" (heap). Купа може бути *керованою* і *некерованою*. Якщо компілятор здатний визначити розмір пам'яті під оголошену змінну, то він виділяє для неї місце в статичній пам'яті (наприклад, під змінну типу `int` він виділить 4 байта, а під масив типу `int` з `10` елементів він виділить `40` байтів).

Якщо ж компілятор не в змозі визначити розмір пам'яті під оголошену змінну, то він вимагатиме від програміста помістити таку змінну в купі — в динамічній пам'яті. До речі, якщо програміст сам хоче працювати з динамічною пам'яттю (навіть коли компілятор може помістити змінну в статичній пам'яті), то мова це йому дозволяє. Для простих змінних при цьому використовуються:

- функція `malloc()`, яка виділяє область пам'яті в динамічній області і повертає покажчик на цю область;

- функція `free()`, яка звільняє зайняту змінної область і передає звільнену пам'ять в загальне користування.

Якщо ж робота відбувається з об'єктами, то тут використовуються оператори `new` — аналог `malloc()` і `delete` — аналог `free()`.

Зазначені функції і аналогічні їм оператори працюють з *некерованою купою*. З керованою купою працює оператор `gcnew` і немає необхідності звільняти самому пам'ять від об'єкта, оскільки в цьому випадку працює так звана *автоматична збірка сміття*: коли об'єкт стає непотрібним, пам'ять від нього звільняється.

Щоб працювати з такою купою, треба перейти в режим CLR. Що стосується питання "чому?", то очевидно, що компілятор розміщує змінну типу класу, оголошеного нами, в статичній пам'яті.

Але наведемо приклад цієї програми, в якій використовується (за нашим бажанням) динамічна пам'ять (ми, природно, наводимо тільки тіло функції `main()`, де це відбивається).

Лістинг 2.6

```
int main()
{
	A *min = (A*) new A(); //створення примірників класів A, B
	B *max = (B*) new B();
	min->a = 10; //Работа з елементами класу A з секції public
	min->b = 20;
	int x1 = min->f1(min->a, min->b);
	int x2 = max->f2(10); //Работа з елементом класу B
	printf("x1=%d\nx2=%d\n", x1, x2);
	_getch();
	delete min;
	delete max;
}
```

У даному випадку оператор `new` розміщує об'єкт в купі, видає адресу об'єкта, а конструктор ініціалізує обсяг.

В наступному прикладі створимо клас, членами якого будуть вироби, що складаються з деталей і їх вартостей, а також методи, перший з яких присвоює значення виробу, деталі, а також їх вартості через свої параметри, а другий виводить на екран значення, присвоєні першим методом. Текст програми приведений в лістингу 2.7, результат роботи програми представлений на рис. 2.2.

Лістинг 2.7

```
#include "stdafx.h"
#include <stdio.h> //для printf()
#include <conio.h> //для _getch()

class produce //початок визначення класу
{
private:
	//Поля класу:
	int modelnumber; //номер виробу
	int partnumber; //номер деталі
	float cost;	//вартість деталі
public:
	//Установка даних за допомогою метода
	//Присвоює даним класу значення своїх параметрів:
	void setpart(int mn, int pn, float c)
	{
		modelnumber = mn;
		partnumber = pn;
		cost = c;
	}

	void show() //вивід даних
	{
		printf("The Number of the Model is %d\n", modelnumber);
		printf("The Number of the Part is %d\n", partnumber);
		printf("The Cost of the Part is %.2f\n", cost);
	}
}; //кінець опису класу

//Обробка класу в головній програмі
int main()
{
	produce izd; //визначення об'єкту з класу (примірник класу)
	izd.setpart(100, 200, 250.5); //виклик методу класу
	izd.show(); //виведення даних
	_getch();
}
```

Цей невеликий створений нами клас дозволяє виводити на екран характеристики виробу, описаного в ньому.

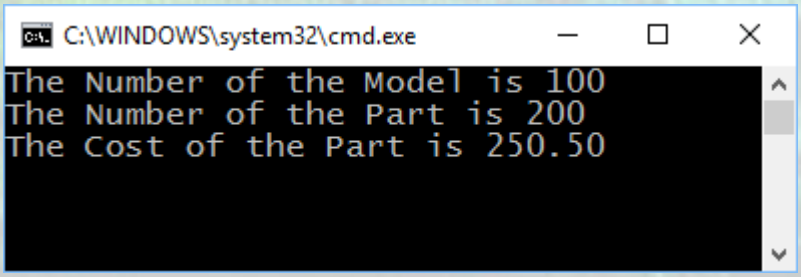


Рисунок 2.2 – Результат роботи програми з лістингу 6.7

Використаємо клас, створений в попередньому прикладі (лістинг 2.7), для створення нового класу — спадкоємця класу з лістингу 2.7. Новий клас повинен буде надавати додаткову характеристику виробу — його форму. Приклад програми наведено у лістингу 2.8, результат роботи програми — на рис. 2.3.

Лістинг 2.8

```
#include "stdafx.h"
#include <stdio.h> //для printf()
#include <conio.h> //для _getch()

class produce //початок визначення класу
{
private:
	//Поля класу:
	int modelnumber; //номер виробу
	int partnumber; //номер деталі
	float cost;	//вартість деталі
public:
	//Установка даних за допомогою метода
	//Присвоює даним класу значення своїх параметрів:
	void setpart(int mn, int pn, float c)//функція-член класу
	{
		modelnumber = mn;
		partnumber = pn;
		cost = c;
	}

	void show() //вивід даних
	{
		printf("The Number of the Model is %d\n", modelnumber);
		printf("The Number of the Part is %d\n", partnumber);
		printf("The Cost of the Part is %.2f\n", cost);
	}
}; //кінець опису класу

//Об'явлення класу-нащадка з новими членами:
class MoreProduce : public produce
{
public:
	char *ProduceForm; //опис форми виробу

	void FormDecl(char *s)
	{
		ProduceForm = s;
	}
```

```
        void show1()
        {
                printf("The Produce Form is %s\n", ProduceForm);
        }
};

//Обробка класу в головній програмі:
int main()
{
        MoreProduce newizd;
        newizd.setpart(100, 200, 250.5);
        newizd.FormDecl("Square");
        newizd.show();
        newizd.show1();
        _getch();
}
```

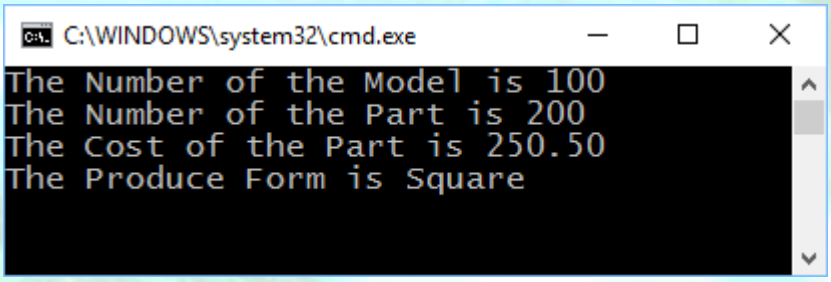


Рисунок 2.3 – Результат роботи програми з лістингу 6.8

2.3.2 Керовані класи CLR

Поряд зі звичайними класами і структурами (їх ще називають native-класи та native-структури або *рідні* класи та структури), існують класи і структури для роботи в середовищі CLR — Common Language Runtime (їх ще називають managed-класи і managed-структури або *керовані* класи та структури). Ці конструкції будуть розташовуватися у виділеній середовищем CLR пам'яті і тому програми з застосуванням цього типу структур повинні компілюватися з ключем /clr. Додатки типу Windows Form відповідають цій умові. Нагадаємо, що об'єкти, які потрапляють в пам'ять під управлінням CLR, вимагають у програмі спеціального виділення пам'яті, так як вони повинні потрапити в дійсності в динамічну пам'ять. Покажчики на таку керовану пам'ять позначаються символом ^ і називаються *дескрипторами*. Виділяють таку пам'ять з допомогою оператора gcnew.

Структури і класи для роботи в режимі CLR можуть бути типу посилання (перед ім'ям класу або структури вказується кваліфікатор ref) і типу значення

(перед ім'ям класу або структури вказується кваліфікатор value). При оголошенні такої конструкції першим кваліфікатором йде кваліфікатор доступу до даної конструкції. Для класу за замовчуванням йде кваліфікатор private, а для структури — public, що також має місце і для прототипів цих конструкцій в режимі native. У чому сенс цих конструкцій?

Наприклад, візьмемо класи:

```
ref class MyClass
{
public:
	int m_i;
};

value class MyClassV
{
public:
	int m_i;
};
```

Це звичайні класи, лише атрибут ref вказує, що його клас потрібно розташовувати у керованій купі.

Наприклад:

```
MyClass ^mc = gcnew MyClass();
```

Якщо ж спробувати його розмістити в native-купі, то компілятор виведе помилку. А для класу з атрибутом value компілятор дозволяє це робити, тобто оператор:

```
MyClassV *vv = new MyClassV();
```

спрацьовує.

Розглянемо наступний приклад.

Лістинг 2.9

```
int main() //Головна функція
{
	MyClass ^mc = gcnew MyClass();//працює
	// MyClass *mc=newMyClass();//не працює
	mc->m_i = 111; //присвоїли значення члену класу
	int mci = mc->m_i; //дістали значення з класу
	//MyClassV ^vv=gcnew MyClassV(); //працює
	MyClassV *vv = new MyClassV(); // працює
```

```
	vv->m_i = 222;
	int mv = vv->m_i;
	Console::WriteLine("mci=" + mci + " mv=" + mv);
	delete vv;
}
```

Результатом роботи даної програми буде:

```
mci=111 mv=222
```

Виявляється, що можна посилатися з керованого типу (managed) на рідний тип (native). Наприклад, функція в керованому класі або структурі може приймати некерований, рідний (native) параметр (наприклад, struct). При цьому якщо керований тип і функція мають атрибут доступу public, то і рідний тип повинен мати атрибут public.

Лістинг 2.10

```
#include "stdafx.h"

using namespace System;

public struct Natv //native-тип
{
	int i;
};

public ref struct Mang //managed-тип
{
	// функція, в якої параметр nn має native-тип Natv:
	int f(Natv nn)
	{
		nn.i++;
		return(nn.i);
	}
};

int main() //Головна функція
{
	Mang ^r = gcnew Mang;
	Natv n;
	n.i = 333;
	int ni = r->f(n);
	Console::WriteLine("ni=" + ni);
}
```

Результатом роботи даної програми буде:

```
ni=334
```

В даному прикладі в керованій структурі Mang визначена функція-член f, яка має параметр nn типу рідної структури Natv.

```
int f(Natv nn)
```

2.3.3 Особливості класів в C++/CLI

Мова програмування C++/CLI має власні типи структур і класів. Фактично C++/CLI дозволяє визначати по два різних види користувальницьких структур і класів, які мають різні характеристики: `value struct` (структури значення) і `value class` (класи значення), а також `ref struct` (структури посилання) і `ref class` (класи посилання). Кожна з комбінацій двох слів - `value struct`, `ref struct`, value class і `ref class` - є ключовим словом і відрізняється від ключових слів `struct` і `class`; слова `value` і `ref` самі по собі не є ключовими. Як і в «рідній» C++, єдина відмінність між класами і структурами в мові C++/CLI полягає в тому, що члени структур за замовчуванням відкриті (`public`), а члени класів за замовчуванням закриті (`private`). Одна суттєва різниця між класами значень (і структурами значень), з одного боку, і класами посилань (і структурами посилань), з іншого, полягає в тому, що змінні типів значень містять свої власні дані, тоді як змінні типів посилань повинні бути дескрипторами, а тому вони містять лише адреси.

Функції-члени класів C++/CLI не можуть бути оголошені константними. Інша відмінність від «рідної» C++ полягає в тому, що покажчик `this` в нестатичних функціях-членах класу значень типу T є внутрішнім покажчиком типу `interior_ptr<T>`, в той час як покажчик `this` в класі посиланні T - це дескриптор типу T^. Слід мати це на увазі при поверненні покажчика `this` з функцій C++/CLI або при збереженні його в локальній змінній. Існують три інших обмеження, які відносяться і до класів значень, і до класів посилань.

- Клас значень або клас посилань не може містити поля, що є масивами «рідної» C++ або типами класів «рідної» C++.
- Дружні функції не дозволені.
- Клас значень або клас посилань не може мати членів, що представляють собою бітові поля.

Ви вже знаєте, що імена базових типів, такі як `int` і `double`, є скороченнями для типів класів значень в програмах CLR. Коли ви оголошуєте елемент даних типу класу значень, пам'ять для нього виділяється в стеці, однак можна створювати об'єкти класів значень і в розподіленій пам'яті, застосовуючи оператор `gcnew`, - в цьому випадку змінна, яка використовується для звернення

до об'єкту класу значень, повинна бути дескриптором. Відповідний приклад наведено нижче.

```
double pi(3.142);           // Pi зберігається в стеку
int^ lucky(gcnew int(7));   // Lucky - дескриптор, і 7 зберігається
                            //в розподіленій пам'яті
double^ two(2.0);           // Two - дескриптор, і 2.0 зберігається
                            //в розподіленій пам'яті
```

Ви можете використовувати будь-яку з цих змінних в арифметичних виразах, але слід застосовувати оператор * для звернення до значення дескрипторів. Щоб звернутися до значення, можна вчинити так, як показано нижче.

```
Console::WriteLine(L"2pi = {0}", *two * pi);
```

Ви можете записати вираз як `pi ** two` і отримати правильний результат, але краще в таких випадках використовувати дужки і писати `pi * (*two)`, що зробить код ясніше.

2.3.4 Визначення типів класів значень

Ми не будемо розглядати типи структур значень окремо від класів значень, оскільки різниця між ними полягає лише в тому, що члени структур значень за замовчуванням відкриті, а члени класів значень за замовчуванням закриті. Клас значень задуманий як відносно простий тип, що дозволяє визначати нові примітивні типи, які можна використовувати аналогічно базовим типам; але ви не повинні робити цього в повній мірі до тих пір, поки не вивчите тему *перевантаження операторів*. Змінна типу класу значень створюється в стеці і зберігає своє значення безпосередньо, але, як ви вже бачили, можна також застосовувати відстежуваний дескриптор, щоб посилатися на тип класу значень, що зберігається в розподіляє пам'яті CLR.

Розглянемо приклад визначення простого класу значень.

```
// Клас для представлення зросту
value class Height
{
private:
      // Поля для запису зросту у метрах та сантиметрах
      int meter;
      int cntmeter;

public:
      // Створює зріст зі значення в сантиметрах
```

```
        Height(int cnms)
        {
            meter = cnms / 100;
            cntmeter = cnms % 100;
        }

        // Створює зріст зі значення в метрах та сантиметрах
        Height(int mtrs, int cnms) : meter(mtrs), cntmeter(cnms) {}
};
```

Це визначає тип класу значень Height. У нього є два закритих поля - обидва типи int, які зберігають значення зросту в метрах і сантиметрах. У класі передбачено два конструктора - один для створення об'єкта Height з кількості сантиметрів, переданих в аргументі, а другий - для створення об'єкта Height на підставі специфікації метрів і сантиметрів. Останній повинен перевіряти, що передано число сантиметрів менше 100. Змінну типу Height можна створити так, як показано нижче.

```
Height tall = Height(1, 83); // Зріст 1 метр 83 сантиметри
```

Цей оператор створює змінну tall, яка містить об'єкт Height, що представляє 1 метр 83 сантиметри; цей об'єкт створюється викликом конструктора з двома параметрами.

```
Height baseHeight;
```

Цей оператор створить змінну baseHeight, яка буде автоматично ініціалізована нульовим зростом. Клас Height не має конструктора без аргументів, і оскільки це клас значень, ви не можете визначити його самостійно у визначенні класу. Тому в клас значень буде автоматично включений конструктор без аргументів, який ініціалізує всі значення полів еквівалентом нуля, а поля-дескриптори - nullptr, і ви не можете замінити цей неявний конструктор власною версією. Саме цей стандартний конструктор і буде застосовуватися для створення значення baseHeight.

Для вмісту класу значень є ще пара обмежень.

- В визначення класу значень не можна включати конструктор копіювання.
- У класі значень не можна перевизначити оператор присвоєння.

Об'єкти класів значень завжди копіюються простим копіюванням полів, і привласнення одного об'єкта такого класу іншим виконується так само. Класи значень призначені для подання простих об'єктів, що містять в собі невеликий обсяг даних, тому для представлення об'єктів, що не задовольняють таким характеристикам, або для яких обмеження класів значень створюють проблеми, необхідно використовувати класи посилань.

Створимо в консольному додатку клас типу значення Height.

```
value class Height
{
private:
	// Поля для запису зросту у метрах та сантиметрах
	int meter;
	int cntmeter;

public:
	// Створює зріст зі значення в сантиметрах
	Height(int cnms)
	{
		meter = cnms / 100;
		cntmeter = cnms % 100;
	}

	// Створює зріст зі значення в метрах та сантиметрах
	Height(int mtrs, int cnms) : meter(mtrs), cntmeter(cnms) {}
};
```

Оголосимо три змінні типу створеного класу значень Height.

```
Height myHeight(Height(1, 73));
Height^ yourHeight(Height(166));
Height hisHeight(*yourHeight);
```

Перша змінна має тип Height, тому об'єкт, що представляє зріст в 1 метр і 73 сантиметри, розміщується в стеку. Друга змінна - дескриптор типу Height^, тому об'єкт, що представляє зріст в 1 метр і 66 сантиметрів, створюється в розподіленій пам'яті CLR. Третя змінна - ще одна стековая змінна, яка є копією об'єкта, на який посилається об'єкт yourHeight. Оскільки об'єкт yourHeight - дескриптор, потрібно звернення до його значення, щоб присвоїти його змінній hisHeight, в результаті чого об'єкт hisHeight містить дублікат об'єкта, на який посилається yourHeight. Змінні класу значень містять унікальні об'єкти, тому дві такі змінні не можуть посилатися на один і той же об'єкт; привласнення однієї змінної типу класу значень іншій такій змінній завжди має на увазі копіювання. Зрозуміло, кілька дескрипторів можуть посилатися на єдиний об'єкт і привласнювати значення одного дескриптора іншому, просто копіюючи адресу (або nullptr), так що обидва об'єкти посилаються на один і той же об'єкт.

Для виведення значень змінних на екран скористаємося стандартною функцією WriteLine.

```
Console::WriteLine(L"Мій зріст складає {0}", myHeight);
Console::WriteLine(L"Твій зріст складає {0}", yourHeight);
```

```
Console::WriteLine(L"Його зріст складає {0}", hisHeight);
```

Виконання цієї програми виводить на екран наступний результат.

```
Мій зріст складає Height
Твій зріст складає Height
Його зріст складає Height
```

Виведення менш інформативне, ніж ми могли сподіватися, але ми повернемося до цього трохи пізніше.

Виведення виконується двома викликами функції `Console::WriteLine()`. На жаль, при цьому не виводяться конкретні значення об'єктів, а просто ім'я класу. Як це виправити? Взагалі кажучи, було б занадто оптимістично очікувати, що будуть виведені значення, - врешті-решт, звідки компілятор може знати, як їх представити? Об'єкт `Height` містить два значення: яке з них потрібно показати? Клас повинен мати спосіб представлення значення в цьому контексті.

2.3.5 Функція класу ToString()

Кожен клас C++/CLI, який ви визначаєте, включає функцію `ToString()`. Ця функція повертає дескриптор рядка, що представляє об'єкт класу. Компілятор передбачає виклик функції `ToString()` для об'єкта щоразу, коли очікується строкове представлення вмісту об'єкта, але при необхідності можете викликати її і явно. Наприклад, можна написати так, як показано нижче.

```
double pi(3.142);
Console::WriteLine(pi.ToString());
```

На екран буде виведено значення `pi` у вигляді рядка, а викликана тут функція `ToString()` визначена в класі `System::Double`. Звичайно, той же результат ви отримаєте і без явного виклику функції `ToString()`.

Стандартна версія функції `ToString()`, яка використовується в класі `Height`, просто виводить ім'я класу, оскільки немає способу дізнатися заздалегідь, яке значення має бути повернуто у вигляді рядка для об'єкта вашого класу. Щоб отримати відповідне значення для виведення функцією `Console::WriteLine()` в попередньому прикладі, потрібно додати в клас `Height` власну функцію `ToString()`, що представляє необхідне значення об'єкта.

Нижче показано, як буде виглядати клас з функцією `ToString()`.

```
// Клас для представлення зросту
value class Height
```

```
{
private:
	// Поля для запису зросту у метрах та сантиметрах
	int meter;
	int cntmeter;

public:
	// Створює зріст зі значення в сантиметрах
	Height(int cnms)
	{
		meter = cnms / 100;
		cntmeter = cnms % 100;
	}

	// Створює зріст зі значення в метрах та сантиметрах
	Height(int mtrs, int cnms): meter(mtrs), cntmeter(cnms) {}

	// Створює рядок для представлення об'єкту
	virtual String^ ToString() override
	{
		return meter + L" метр(ів) " + cntmeter + L" сантиметри(ів)";
	}
};
```

Комбінація ключового слова `virtual` перед типом значення, що повертає `ToString()`, і ключового слова `override`, за яким слідує список параметрів функції, вказує, що ця версія функції `ToString()` перевизначає версію, надану компілятором класу за замовчуванням. Наша нова версія функції `ToString()` виводить тепер рядок, який має зріст в метрах і сантиметрах. Якщо ви додасте цю функцію в визначення класу з попереднього прикладу, то отримаєте наступне виведення після компіляції і виконання програми.

```
Мій зріст складає 1 метр(ів) 73 сантиметри(ів)
Твій зріст складає 1 метр(ів) 66 сантиметри(ів)
Його зріст складає 1 метр(ів) 66 сантиметри(ів)
```

Це більше схоже на те, що ви очікували отримати. Як видно в цьому виведенні, функція `WriteLine()` цілком успішно справляється з об'єктом з розподіленої пам'яті CLR, на який посилається дескриптор `yourHeight`, так само, як і з об'єктами `myHeight` і `hisHeight`, які створені в стеці.

2.3.6 Визначення класів посилань

Клас посилань за своїми можливостями можна порівняти з класом «рідної» C++ і не має обмежень, які притаманні класу значень. Але на відміну від класу «рідної» C++ клас посилань не має стандартного конструктора копіювання

або оператора присвоєння за умовчанням. Якщо ваш клас повинен підтримувати будь-яку з цих операцій, необхідно явно додати відповідну функцію для її реалізації.

Клас посилань визначається з використанням ключового слова `ref class` - слова в цьому поєднанні, розділені одним або декількома пробілами, являють собою єдине ключове слово. Розглянемо клас Box визначений як клас посилань.

```
ref class Box
{
public:
      // Конструктор без аргументів що встановлює значення за
замовчуванням
      Box(): Length(1.0), Width(1.0), Height(1.0)
      {
            Console::WriteLine(L"Викликаний конструктор без аргументів.");
      }

      // Визначення конструктора зі списком ініціалізації
      Box(double lv, double bv, double hv): Length(lv), Width(bv),
            Height(hv)
      {
            Console::WriteLine(L"Викликано конструктор.");
      }

      // Функція розрахунку об'єму ящика
      double Volume()
      {
            return Length*Width*Height;
      }

private:
      double Length;         // Довжина ящика в сантиметрах
      double Width;          // Ширина ящика в сантиметрах
      double Height;         // Висота ящика в сантиметрах
};
```

Ви не можете задати значення за замовчуванням для параметрів функцій і конструктора в класах C++/CLI, тому повинні додати в клас Box конструктор без аргументів. Цей конструктор просто ініціалізує три закритих поля значеннями `1.0`.

Нижче наведений приклад використання класу Box.

```
Box^ aBox;              // Дескриптор типу Box^
Box^ newBox = gcnew Box(10, 15, 20);
aBox = gcnew Box;       // Ініціалізувати Box за замовчуванням
Console::WriteLine(L"Об'єм ящика за замовчуванням {0} см куб.",
      aBox->Volume());
```

```
Console::WriteLine(L"Об'єм нового ящика {0} см куб.", newBox->Volume());
```

Результат роботи програми буде наступний.

```
Викликано конструктор.
Викликаний конструктор без аргументів.
Об'єм ящика за замовчуванням 1 см куб.
Об'єм нового ящика 3000 см куб.
```

Третій оператор у функції `main()` створює дескриптор об’єкту Box.

```
Box^ aBox;              // Дескриптор типу Box^
```

Цей оператор не створює ніяких об'єктів, створюється тільки відстежуваний дескриптор - aBox. Змінна aBox ініціалізується за замовчуванням значенням `nullptr`, тому поки вона ні на що не вказує. На відміну від цього змінна класу значення завжди містить об'єкт.

Наступний оператор створює дескриптор нового об'єкта Box.

```
Box^ newBox = gcnew Box(10, 15, 20);
```

Конструктор, який одержує три аргументи, викликається для створення в розподіленій пам'яті об'єкта Box, і його адреса зберігається в дескрипторі newBox. Як ви знаєте, об'єкти типів класів посилань завжди створюються в розподіленій пам'яті CLR і завжди доступні тільки через дескриптор.

Ви створюєте об'єкт Box, викликаючи конструктор без аргументів, і зберігаєте його адресу в змінній aBox.

```
aBox = gcnew Box;           // Ініціалізувати Box за замовчуванням
```

Всі поля цього об'єкта - `Length`, `Width` та `Height` - встановлені в значення `1.0`.

І нарешті, виводяться об’єми двох щойно створених об'єктів Box.

```
Console::WriteLine(L"Об'єм ящика за замовчуванням {0} см куб.",
    aBox->Volume());
Console::WriteLine(L"Об'єм нового ящика {0} см куб.", newBox->Volume());
```

Оскільки aBox і newBox - дескриптори, при виконанні функції `Volume()` для об'єктів, на які вони вказують, використовується оператор `->`.

2.3.7 Визначення конструктора копіювання для класів посилань

Малоймовірно, що це доведеться робити часто, але якщо ви передаєте об'єкти типу класу посилань в функцію за значенням, то повинні реалізувати

відкритий конструктор копіювання. Така ситуація може виникнути з реалізацією стандартної бібліотеки шаблонів для CLR. Параметр для конструктора копіювання повинен бути константним посиланням, так що необхідно визначити конструктор копіювання для класу Box наступним чином.

```
// Визначення конструктора копіювання класу при передачі його
//об'єкта у функцію за значенням
Box(const Box% box): Length(box.Length), Width(box.Width),
Height(box.Height) {}
```

У загальному випадку форма конструктора копіювання для класу посилань T, що дозволяє передавати об'єкт посилання типу T в функцію за значенням, наведена нижче.

```
T (const T% t)
{
// Код, який реалізує копіювання ...
}
```

Іноді також може знадобитися реалізувати конструктор копіювання, який одержує аргумент - дескриптор. Як це можна зробити для класу Box, показано нижче.

```
//Визначення конструктора копіювання класу при передачі його
//об'єкта у функцію за посиланням
Box(const Box^ box): Length(box -> Length), Width(box -> Width),
      Height(box -> Height) {}
```

Як бачите, між цією і попередньою версією є невелика відмінність.

2.3.8 Властивості класів

Властивість (property) - це член або класу посилань, або класу значення, до якого ви звертаєтеся як до поля, але яке, однак, полем не є. Головна відмінність між властивістю і полем полягає в тому, що ім'я поля посилається на місце в пам'яті, де зберігається об'єкт даних, в той час як ім'я властивості викликає функцію. Властивість має *функції доступу* `get()` та `set()` - відповідно для отримання і установки значення. Так що коли ви використовуєте ім'я властивості, щоб прочитати його значення, внутрішньо викликається функція `get()` для властивості, а коли ви застосовуєте ім'я властивості в лівій частині оператора присвоєння, викликається функція `set()`. Якщо властивість представляє тільки визначення функції `get()`, вона називається *властивістю тільки для читання*, тому що функція `set()` недоступна для установки значення

властивості. Властивість може мати тільки функцію `set()`, в цьому випадку вона буде *властивістю тільки для запису*.

Клас може містити два різних види властивостей: *скалярні* і *індексовані*. Скалярні властивості являють собою єдине значення, доступне через ім'я властивості, а індексовані властивості - це набори значень, до яких ви звертаєтеся, вказуючи індекс в квадратних дужках, наступних за ім'ям властивості. Клас `String` має скалярну властивість `Length`, що представляє кількості символів в рядку, і для об'єкта `str` типу `String` ви звертаєтеся до властивості `Length` за допомогою виразу `str -> Length`, тому що `str` - дескриптор. Звичайно, для звернення до властивості `MyProp` об'єкта класу значення, що зберігається в змінній `val`, потрібно використовувати вираз `val.MyProp`, як при доступі до поля. Властивість `Length` рядка - приклад властивості тільки для читання, оскільки для неї не визначена функція `set()`, - ви не можете встановити довжину рядка, так як об'єкт `String` не змінюється. Клас `String` також надає доступ до індивідуальних символів рядка у вигляді індексованої властивості. Для рядкового дескриптора `str` можете звернутися до третьої індексованої властивості за допомогою виразу `str[2]`, який відповідає третьому символу рядка.

Властивості можуть бути асоційовані з конкретним об'єктом - в цьому випадку вони описуються як властивості екземпляра. Властивість `Length` об'єкта `String` - приклад властивості екземпляра. Ви можете також вказати властивість як `static`, і в цьому випадку властивість асоційована з класом в цілому, а її значення є загальним для всіх об'єктів класу. Розглянемо властивості трохи докладніше.

Приклад програмної реалізації

Приклад створення класу типу значення в консольного додатку CLR

Створимо в консольному додатку клас типу значення `Height`.

```
// Клас для представлення зросту
value class Height
{
private:
	// Поля для запису зросту у метрах та сантиметрах
	int meter;
	int cntmeter;

public:
```

```
        // Створює зріст зі значення в сантиметрах
        Height(int cnms)
        {
            meter = cnms / 100;
            cntmeter = cnms % 100;
        }

        // Створює зріст зі значення в метрах та сантиметрах
        Height(int mtrs, int cnms) : meter(mtrs), cntmeter(cnms) {}

        // Створює рядок для представлення об'єкту
        virtual String^ ToString() override
        {
            return meter + L" метр(ів) " + cntmeter + L" сантиметри(ів)";
        }

};
```

Надалі створюємо три змінні типу Height.

```
Height myHeight(Height(1, 73));
Height^ yourHeight(Height(166));
Height hisHeight(*yourHeight);
```

Значення створених змінних виводимо на екран за допомогою функції WriteLine.

```
Console::WriteLine(L"Мій зріст складає {0}", myHeight);
Console::WriteLine(L"Твій зріст складає {0}", yourHeight);
Console::WriteLine(L"Його зріст складає {0}", hisHeight);
```

Для можливості відображення української мови в програмі застосовані функції SetConsoleCP() та SetConsoleOutputCP() для зміни таблиці кодування на cp1251.

```
SetConsoleCP(1251);  //Встановлення кодування для введення укр. мови
SetConsoleOutputCP(1251); //Встановлення кодування для виведення укр. мови
```

Для роботи даних функцій потрібно підключити заголовочний файл windows.h.

```
#include <windows.h>
```

Також застосована функція system("pause") для затримки консольного вікна по завершенню роботи програми.

```
system("pause"); //Затримка консольного вікна до натискання клавіши
```

Комбінація ключового слова virtual перед типом значення, що повертає ToString(), і ключового слова override, за яким слідує список параметрів функції, вказує, що ця версія функції ToString() перевизначає версію, надану компілятором класу за замовчуванням. Наша нова версія функції ToString() виводить тепер рядок, який має зріст в метрах і сантиметрах.

Виконання цієї програми виводить на екран наступний результат.

```
Мій зріст складає 1 метр(ів) 73 сантиметри(ів)
Твій зріст складає 1 метр(ів) 66 сантиметри(ів)
Його зріст складає 1 метр(ів) 66 сантиметри(ів)
```

Як видно в цьому виведенні, функція WriteLine() цілком успішно справляється з об'єктом з розподіленої пам'яті CLR, на який посилається дескриптор yourHeight, так само, як і з об'єктами myHeight і hisHeight, які створені в стеці.

Повний текст програмного модуля наведений в нижче.

```
// ClassValue.cpp: главный файл проекта.
// Визначення і використання типу класу значень

#include "stdafx.h"
#include "windows.h"

using namespace System;

// Клас для представлення зросту
value class Height
{
private:
      // Поля для запису зросту у метрах та сантиметрах
      int meter;
      int cntmeter;

public:
      // Створює зріст зі значення в сантиметрах
      Height(int cnms)
```

```
        {
            meter = cnms / 100;
            cntmeter = cnms % 100;
        }

        // Створює зріст зі значення в метрах та сантиметрах
        Height(int mtrs, int cnms) : meter(mtrs), cntmeter(cnms) {}

        // Створює рядок для представлення об'єкту
        virtual String^ ToString() override
        {
            return meter + L" метр(ів) " + cntmeter + L" сантиметри(ів)";
        }

};

int main(array<System::String ^> ^args)
{
        //Встановлення кодування для введення укр. мови
        SetConsoleCP(1251);
        //Встановлення кодування для виведення укр. мови
        SetConsoleOutputCP(1251);

        Height myHeight(Height(1, 73));
        Height^ yourHeight(Height(166));
        Height hisHeight(*yourHeight);

        Console::WriteLine(L"Мій зріст складає {0}", myHeight);
        Console::WriteLine(L"Твій зріст складає {0}", yourHeight);
        Console::WriteLine(L"Його зріст складає {0}", hisHeight);

        system("pause"); //Затримка консольного вікна до натискання клавіши

        return 0;
}
```

Приклад створення класу типу посилання в консольного додатку CLR

Створимо в консольному додатку клас типу посилання Box.

```
ref class Box
{
```

```
public:
	// Конструктор без аргументів що встановлює значення
	//за замовчуванням
	Box() : Length(1.0), Width(1.0), Height(1.0)
	{
		Console::WriteLine(L"Викликаний конструктор без аргументів.");
	}

	// Визначення конструктора зі списком ініціалізації
	Box(double lv, double bv, double hv): Length(lv), Width(bv),
		Height(hv)
	{
		Console::WriteLine(L"Викликано конструктор.");
	}

	// Функція розрахунку об'єму ящика
	double Volume()
	{
		return Length*Width*Height;
	}

private:
	double Length;		// Довжина ящика в сантиметрах
	double Width;			// Ширина ящика в сантиметрах
	double Height;		// Висота ящика в сантиметрах
};
```

Оголосимо змінну типу Box.

```
Box^ aBox;			// Дескриптор типу Box^
```

Створимо об’єкт класу Box та ініціалізуємо його значенням за замовчуванням:

```
aBox = gcnew Box;			// Ініціалізувати Box за замовчуванням
```

Створимо об’єкт класу Box та ініціалізуємо його потрібними значеннями:

```
Box^ newBox = gcnew Box(10, 15, 20);
```

Виведемо на екран розраховане значення функції Volume класу Box для двох створених примірників класу, які були ініціалізовані значенням за замовчуванням та вказаним набором параметрів:

```
Console::WriteLine(L"Об'єм ящика за замовчуванням {0} см куб.",
      aBox->Volume());
Console::WriteLine(L"Об'єм нового ящика {0} см куб.", newBox->Volume());
```

Результат роботи програми буде наступний.

```
Викликано конструктор.
Викликаний конструктор без аргументів.
Об'єм ящика за замовчуванням 1 см куб.
Об'єм нового ящика 3000 см куб.
```

Повний текст програмного модуля наведений в нижче.

```
// ClassRef.cpp: главный файл проекта.
// Використання класу посилань Box

#include "stdafx.h"
#include "windows.h"

using namespace System;

ref class Box
{
public:
      // Конструктор без аргументів що встановлює значення
      //за замовчуванням
      Box() : Length(1.0), Width(1.0), Height(1.0)
      {
            Console::WriteLine(L"Викликаний конструктор без аргументів.");
      }

      // Визначення конструктора зі списком ініціалізації
      Box(double lv, double bv, double hv): Length(lv), Width(bv),
            Height(hv)
      {
            Console::WriteLine(L"Викликано конструктор.");
      }

      // Функція розрахунку об'єму ящика
```

```
        double Volume()
        {
            return Length*Width*Height;
        }

private:
        double Length;        // Довжина ящика в сантиметрах
        double Width;         // Ширина ящика в сантиметрах
        double Height;        // Висота ящика в сантиметрах
};

int main(array<System::String^>^ args)
{
        //Встановлення кодування для введення укр. мови
        SetConsoleCP(1251);
        //Встановлення кодування для виведення укр. мови
        SetConsoleOutputCP(1251);
        Box^ aBox;            // Дескриптор типу Box^
        Box^ newBox = gcnew Box(10, 15, 20);
        aBox = gcnew Box;     // Ініціалізувати Box за замовчуванням
        Console::WriteLine(L"Об'єм ящика за замовчуванням {0} см куб.",
            aBox->Volume());
        Console::WriteLine(L"Об'єм нового ящика {0} см куб.",
            newBox->Volume());
        return 0;
}
```

2.4 Округлення числових значень

Для виконання операції округлення використовуються методи `Math::Round`, `Math::Ceiling` або `Math::Floor`.

2.4.1 Метод `Math::Round`

Метод `Math::Round` округлює значення до найближчого цілого або вказаної кількості десяткових знаків. Перевантажені версії метода `Math::Round` наведені в табл. 2.1.

Таблиця 2.1 – Список перевантажень методу `Math::Round`

Ім'я методу	Опис
`Round(Decimal)`	Коли число знаходиться посередині між двома іншими числами, воно округляється до найближчого більшого за модулем числа
`Round(Decimal, Int32)`	Округлює десяткове значення до зазначеного числа дрібних розрядів
`Round(Decimal, Int32, MidpointRounding)`	Округлює десяткове значення до зазначеного числа дробових розрядів. Параметр задає правило округлення значення, якщо воно знаходиться рівно посередині між двома числами
`Round(Decimal, MidpointRounding)`	Округлює десяткове значення до найближчого цілого. Параметр задає правило округлення значення, якщо воно знаходиться рівно посередині між двома числами
`Round(Double)`	Округлює задане число з плаваючою комою подвійної точності до найближчого цілого
`Round(Double, Int32)`	Округлює значення подвійної точності з плаваючою комою до заданої кількості дробових розрядів
`Round(Double, Int32, MidpointRounding)`	Округлює значення подвійної точності з плаваючою комою до заданої кількості дробових розрядів. Параметр задає правило округлення значення, якщо воно знаходиться рівно посередині між двома числами
`Round(Double, MidpointRounding)`	Округлює задане значення числа подвійної точності з плаваючою комою до найближчого цілого. Параметр задає правило округлення значення, якщо воно знаходиться рівно посередині між двома числами

Параметр `MidpointRounding` з простору імен `System` задає спосіб обробки чисел, які рівновіддалені від двох сусідніх чисел, в математичних

методах округлення. Можливі значення для класу `System::MidpointRounding` наведені в таблиці 2.2.

Таблиця 2.2 – Значення для класу `MidpointRounding`

Ім'я елементу	Значення
`AwayFromZero`	Коли число знаходиться посередині між двома іншими числами, воно округляється до найближчого більшого за модулем числа
`ToEven`	Коли число знаходиться посередині між двома іншими числами, воно округляється до найближчого парного числа

2.4.2 Метод `Math::Ceiling`

Метод `Math::Ceiling` повертає найменше ціле число, яке більше або дорівнює заданому числу.

Перелік перевантажень методу `Math::Ceiling` наведений в табл. 2.3.

Таблиця 2.3 – Список перевантажень методу `Math::Ceiling`

Ім'я елементу	Значення
`Ceiling(Decimal)`	Повертає найменше ціле число, яке *більше* або дорівнює заданому десятковому числу
`Ceiling(Double)`	Повертає найменше ціле число, яке *більше* або дорівнює заданому числу з плаваючою комою подвійної точності

2.4.3 Метод `Math::Floor`

Метод `Math::Floor` повертає найбільше ціле число, яке менше або дорівнює зазначеному числу.

Перелік перевантажень методу `Math::Floor` наведений в табл. 2.4.

Таблиця 2.4 – Список перевантажень методу `Math::Floor`

Ім'я елементу	Значення
`Floor(Decimal)`	Повертає найбільше ціле число, яке *менше* або дорівнює зазначеному десятковому числу
`Floor(Double)`	Повертає найбільше ціле число, яке *менше* або дорівнює заданому числу подвійної точності з плаваючою комою

2.4.4 Метод `Math::Truncate`

Метод `Math::Truncate` обчислює цілу частину числа.

Перелік перевантажень методу `Math::Truncate` наведений в табл. 2.5.

Таблиця 2.5 – Список перевантажень методу `Math::Truncate`

Ім'я елементу	Значення
`Truncate(Decimal)`	Обчислює цілу частину заданого десяткового числа
`Truncate(Double)`	Обчислює цілу частину заданого числа подвійної точності з плаваючою комою

2.4.5 Метод `Math::Sign`

Метод `Math::Sign` повертає ціле число, яке вказує знак числа. Даний метод має перелік перевантажень для різних числових форматів. Значення, яке повертає метод `Math::Sign`, залежить від переданого числового значення (табл. 2.6).

Таблиця 2.6 – Значення, яке повертає метод `Math::Sign`

Повернене значення	Числове значення
`-1`	Значення числа меньше нуля
`0`	Значення числа дорівнює нулю
`1`	Значення числа більше нуля

2.5 Перетворення числових значень в рядкові методом `ToString`

Форматування - це процес перетворення екземпляра класу, структури або значення перерахування в рядкове представлення. Рядок потім демонструється користувачам або десеріалізуется для подальшого відновлення значення з вихідним типом даних.

Метод `ToString` може бути кількох перевантажених видів (табл. 2.7).

Таблиця 2.7 – Перевантаження методу `Double.ToString`

Метод	Призначення
`ToString()`	перетворює числове значення даного екземпляра в еквівалентне йому рядкове представлення без форматування
`ToString(String)`	перетворює числове значення даного екземпляра в еквівалентне рядкове представлення з використанням зазначеного формату у вигляді текстового рядка
`ToString(IFormatProvider)`	перетворює числове значення даного екземпляра в еквівалентне йому рядкове представлення з використанням зазначених відомостей про особливості форматування для даної мови і регіональних параметрів
`ToString(String, IFormatProvider)`	перетворює числове значення даного екземпляра в еквівалентне йому рядкове представлення з використанням зазначеного формату і відомостей про особливості форматування для даної мови і регіональних параметрів

2.6 Стандартні числові формати рядкових даних

Платформа .NET Framework забезпечує широку підтримку форматування. Рядки стандартних числових форматів служать для форматування стандартних числових типів. Рядок стандартних числових форматів використовує формат Axx, де:

- A - це один буквений символ, який називають *описувачем формату*. Будь-який рядок числового формату, що містить більше однієї літери, включаючи пробіли, інтерпретується як рядок настроювання числового формату.
- xx - це необов'язкове ціле число, яке називають *описувачем точності*. Специфікатор точності знаходиться в діапазоні від 0 до 99 і впливає на число цифр в результаті. Описувач точності управляє кількістю цифр в рядковому поданні числа. Він не округлює саме число. Для виконання операції округлення використовуйте метод Math::Ceiling, Math::Floor або Math::Round.

Якщо описувач точності визначає число цифр дробової частини в підсумковому рядку, то дробова частина округлюється в більшу сторону (тобто з використанням MidpointRounding::AwayFromZero).

У наведеній нижче таблиці описано специфікатори стандартних числових форматів і відображені приклади вихідних даних, отримані за допомогою кожного специфікатора формату (табл. 2.8).

Таблиця 2.8 – Специфікатори стандартних числових форматів

Описувач формату	Назва	Опис	Приклади
"C" або "c"	Валюта (Currency)	Результат: значення у валюті. Підтримується: для всіх числових типів даних. Описувач точності: кількість цифр в дробовій частині. Описувач точності за замовчуванням: визначається NumberFormatInfo::CurrencyDecimalDigits	123.456 ("C", en-US) -> $123.46 123.456 ("C", fr-FR) -> 123,46€ -123.456 ("C3", fr-FR) -> -123,456€

Описувач формату	Назва	Опис	Приклади
"D" або "d"	Десятковий (Decimal)	Результат: цілочисельні цифри з необов'язковим негативним знаком. Підтримується: тільки цілочисельними типами даних. Описувач точності: мінімальна кількість цифр. Описувач точності за замовчуванням: мінімально необхідна кількість цифр.	1234 ("D") -> 1234 -1234 ("D6") -> -001234
"E" або "e"	Експоненційний (науковий) (Exponential)	Результат: експоненційний запис. Підтримується: для всіх числових типів даних. Описувач точності: кількість цифр в дробовій частині. Описувач точності за замовчуванням: 6.	1052.0329112756 ("E", en-US) -> 1.052033E+003 1052.0329112756 ("e", fr-FR) -> 1,052033e+003 1052.0329112756 ("e2", en-US) -> 1.05e+003
"F" або "f"	З фіксованою комою (Fixed-point)	Результат: цифри цілої і дробової частини з необов'язковим негативним знаком. Підтримується: для всіх числових типів даних. Описувач точності: кількість цифр в дробовій частині. Описувач точності за замовчуванням: визначається NumberFormatInfo::NumberDecimalDigits	1234.567 ("F", en-US) -> 1234.57 1234.567 ("F", de-DE) -> 1234,57 1234 ("F1", en-US) -> 1234.0 -1234.56 ("F4", en-US) -> -1234.5600
"G" або "g"	Загальні правила (General)	Результат: найбільш компактний запис з двох варіантів: експоненційного та з фіксованою комою. Підтримується: для всіх числових типів даних. Описувач точності: кількість значущих цифр. Описувач точності за замовчуванням: визначається числовим типом.	-123.456 ("G", en-US) -> -123.456 123.4546 ("G4", sv-SE) -> 123,5 -1.234567890e-25 ("G", en-US) -> -1.23456789E-25
"N" або "n"	Числовий (Number)	Результат: цифри цілої й дробової частин, роздільники груп і роздільник цілої та дробової частин з необов'язковим негативним знаком. Підтримується: для всіх числових типів даних. Описувач точності: бажане число знаків дробової частини. Описувач точності за замовчуванням: визначається NumberFormatInfo::NumberDecimalDigits	1234.567 ("N", en-US) -> 1,234.57 1234.567 ("N", ru-RU) -> 1 234,57 -1234.56 ("N3", en-US) -> -1,234.560 -1234.56 ("N3", ru-RU) -> -1 234,560

Описувач формату	Назва	Опис	Приклади
"P" або "p"	Відсоток (Percent)	Результат: число, помножене на 100 і відображене з символом процента. Підтримується: для всіх числових типів даних. Описувач точності: бажане число знаків дробової частини. Описувач точності за замовчуванням: визначається NumberFormatInfo::PercentDecimalDigits	1 ("P", en-US) -> 100.00 % 1 ("P", fr-FR) -> 100,00 % -0.39678 ("P1", en-US) -> -39.7 % -0.39678 ("P1", fr-FR) -> -39,7 %
"R" або "r"	Зворотне перетворення (Round-trip)	Результат: рядок, що дає при зворотному перетворенні ідентичне число. Підтримується: Single, Double и BigInteger. Описувач точності за замовчуванням: ігнорується.	123456789.12345678 ("R") -> 123456789.12345678 -1234567890.12345678 ("R") -> -1234567890.12345678
"X" або "x"	Шістнадця-тиричний (Hexadecima	Результат: шістнадцятиричний рядок. Підтримується: лише цілочисельними типами даних. Описувач точності: число цифр в результуючому рядку.	255 ("X") -> FF -1 ("x") -> ff 255 ("x4") -> 00ff -1 ("X4") -> 00FF
Будь-який інший	Невідомий описувач	Результат: породження виключення FormatException під час виконання.	

У наведених нижче поясненнях міститься докладна інформація про всі рядкові стандартні числові формати.

2.6.1 Описувач формату грошової одиниці ("C")

При використанні описувача формату грошової одиниці ("C") число перетворюється в рядок, що представляє суму в деякій валюті. Бажана кількість знаків дробової частини у результуючому рядку задається описувачем точності. Якщо описувач точності не заданий, точність за замовчуванням визначається властивістю NumberFormatInfo::CurrencyDecimalDigits.

Якщо форматоване значення містить більше десяткових знаків, ніж задано або можливо за замовчуванням, в результуючому рядку дробове значення округляється. Якщо значення праворуч від заданого числа десяткових знаків більше або дорівнює 5, останній знак в результуючому рядку округляється в сторону від нуля (в більший бік).

Форматування результуючого рядка визначається відомостями про форматування в поточному об'єкті NumberFormatInfo. У таблиці 2.9 представлені властивості NumberFormatInfo, щоб забезпечити управління форматуванням отриманого рядка.

Таблиця 2.9 – Значення властивості NumberFormatInfo

Значення	Опис
CurrencyPositivePattern	Визначає положення символу валюти в позитивних значеннях
CurrencyNegativePattern	Визначає положення символу валюти в негативних значеннях і вказує, як саме подається негативний знак: круглими дужками або властивістю NegativeSign
NegativeSign	Задає негативний знак, який використовується в разі, якщо властивість CurrencyNegativePattern вказує на те, що дужки для заперечення не використовуються
CurrencySymbol	Визначає символ валюти
CurrencyDecimalDigits	Визначає кількість цифр дробової частини в значенні валюти за замовчуванням. Це значення можна перевизначити за допомогою описувача точності
CurrencyDecimalSeparator	Визначає рядок, що розділяє цілу і дробову частини числа
CurrencyGroupSeparator	Визначає рядок, що розділяє групи цифр цілої частини
CurrencyGroupSizes	Визначає число цілочисельних цифр, що входять в групу

У наступному прикладі значення Double форматується за допомогою специфікатор грошового формату.

```
double value = 12345.6789;
Console::WriteLine(value.ToString("C",
     Globalization::CultureInfo::CurrentCulture));
Console::WriteLine(value.ToString("C3",
     Globalization::CultureInfo::CurrentCulture));
// Приклад відобразить наступний результат
// якщо поточним є формат English (United States):
//    $12,345.68
//    $12,345.679
Console::WriteLine(value.ToString("C3",
     Globalization::CultureInfo::CreateSpecificCulture("da-DK")));
// Приклад відобразить наступний результат
//     12.345,679 kr.
```

2.6.2 Описувач десяткового формату ("D")

При використанні описувача десяткового формату ("D") число перетворюється в рядок, що складається з десяткових цифр (0-9); якщо число

негативне, перед ним ставиться негативний знак. Цей формат доступний тільки для цілих типів.

Мінімальна кількість знаків у вихідному рядку задається специфікатором точності. Відсутні знаки в рядку замінюються нулями. Якщо описувач точності не заданий, за замовчуванням використовується мінімальне значення, що дозволяє представити ціле число без нулів на початку.

Форматування результуючого рядка визначається відомостями про форматування в поточному об'єкті NumberFormatInfo. Як показано в таблиці нижче, управління форматуванням результуючого рядка здійснюється за допомогою однієї властивості (табл. 2.10).

Таблиця 2.10 – Значення властивості NumberFormatInfo

Значення	Опис
NegativeSign	Визначає рядок, який вказує, що число є негативним

У наступному прикладі значення Int32 форматується за допомогою описувача десяткового формату.

```
int value;
value = 12345;
Console::WriteLine(value.ToString("D"));
// Відобразиться 12345

Console::WriteLine(value.ToString("D8"));
// Відобразиться 00012345

value = -12345;
Console::WriteLine(value.ToString("D"));
// Відобразиться -12345

Console::WriteLine(value.ToString("D8"));
// Відобразиться -00012345
```

2.6.3 Описувач експоненціального формату ("E")

При використанні описувача експоненціального формату ("E") число перетворюється в рядок виду "-d.ddd ... E + ddd" або "-d.ddd ... e + ddd", де кожен символ "d" позначає цифру (0-9). Якщо число від'ємне, на початку рядка ставиться негативний знак. Перед роздільником цілої і дробової частини завжди стоїть рівно одна цифра.

Необхідна кількість знаків дробової частини задається специфікатором точності. Якщо специфікатор точності відсутній, за замовчуванням число знаків дробової частини дорівнює шести.

Регістр описувача формату задає регістр літери, що стоїть перед експонентою ("E" або "e"). Експонента складається з знака "плюс" або "мінус" і трьох цифр. Відсутні до мінімуму цифри замінюються нулями, якщо це необхідно.

Форматування результуючого рядка визначається відомостями про форматування в поточному об'єкті NumberFormatInfo. У таблиці 2.11 представлені властивості NumberFormatInfo для забезпечення управління форматуванням повернутого рядка.

Таблиця 2.11 – Значення властивості NumberFormatInfo

Значення	Опис
NegativeSign	Визначає рядок, який вказує на те, що число є негативним (як мантиса, так і експонента)
NumberDecimalSeparator	Визначає рядок, що розділяє цілу і дробову частини мантиси
PositiveSign	Визначає рядок, який вказує, що експонента є позитивною

У наступному прикладі значення Double форматується за допомогою описувача експоненціального формату.

```
double value = 12345.6789;
Console::WriteLine(value.ToString("E",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 1.234568E+004

Console::WriteLine(value.ToString("E10",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 1.2345678900E+004

Console::WriteLine(value.ToString("e4",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 1.2346e+004

Console::WriteLine(value.ToString("E",
     Globalization::CultureInfo::CreateSpecificCulture("fr-FR")));
// Відобразиться 1,234568E+004
```

2.6.4 Описувач формату з фіксованою комою ("F")

При використанні специфікатору формату з фіксованою крапкою ("F") число перетворюється в рядок виду "-ddd.ddd ...", де кожне "d" позначає цифру (0-9). Якщо число від'ємне, на початку рядка ставиться негативний знак.

Необхідна кількість знаків дробової частини задається специфікатором точності. Якщо описувач точності відсутній, то використовується чисельна

точність, обумовлена поточним значенням властивості NumberFormatInfo :: NumberDecimalDigits.

Форматування результуючого рядка визначається відомостями про форматування в поточному об'єкті NumberFormatInfo. У таблиці 2.12 представлені властивості об'єкта NumberFormatInfo, щоб забезпечити управління форматуванням результуючого рядка.

Таблиця 2.12 – Значення властивості NumberFormatInfo

Значення	Опис
NegativeSign	Визначає рядок, який вказує на те, що число є негативним
NumberDecimalSeparator	Визначає рядок, що розділяє цілу і дробову частини числа
NumberDecimalDigits	Визначає кількість цифр дробової частини за замовчуванням. Це значення можна перевизначити за допомогою описувача точності

У наступному прикладі значення Double і значення Int32 форматируются за допомогою специфікатор формату з фіксованою крапкою.

```
int integerNumber;
integerNumber = 17843;
Console::WriteLine(integerNumber.ToString("F",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 17843.00

integerNumber = -29541;
Console::WriteLine(integerNumber.ToString("F3",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться -29541.000

double doubleNumber;
doubleNumber = 18934.1879;
Console::WriteLine(doubleNumber.ToString("F",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 18934.19

Console::WriteLine(doubleNumber.ToString("F0",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 18934

doubleNumber = -1898300.1987;
Console::WriteLine(doubleNumber.ToString("F1",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться -1898300.2

Console::WriteLine(doubleNumber.ToString("F3",
     Globalization::CultureInfo::CreateSpecificCulture("es-ES")));
```

```
// Відобразиться -1898300,199
```

2.6.5 Описувач загального формату ("G")

При використанні описувача загального формату ("G") число перетворюється в найбільш короткий з двох варіантів: запис з фіксованою комою або експоненціальний запис. При цьому враховується тип числа і наявність описувача точності. Описувач точності визначає максимальну кількість значущих цифр, які можуть бути використані в результуючому рядку. Якщо описувач точності не заданий або дорівнює нулю, точність визначається типом числа, як показано в таблиці 2.13.

Таблиця 2.13 – Значення точності за замовчуванням числових типів

Числовий тип	Точність за замовчуванням
Byte або SByte	3 знака
Int16 або UInt16	5 знаків
Int32 або UInt32	10 знаків
Int64	19 знаків
UInt64	20 знаків
BigInteger	50 знаків
Single	7 знаків
Double	15 знаків
Decimal	29 знаків

Вигляд з фіксованою комою використовується, якщо експонента результату в експоненційному вигляді довше п'яти знаків, але менше специфікатору точності, в іншому випадку використовується наукове представлення. При необхідності результат містить роздільник цілої та дробової частин; нулі в кінці дробової частини після роздільника відкидаються. Якщо описувач точності заданий і число значущих цифр результату перевершує вказане значення точності, зайві цифри відкидаються округленням.

Проте, якщо число відноситься до типу `Decimal` і описувач точності не заданий, завжди використовується представлення з фіксованою комою, а нулі в кінці, не відкидаються.

Якщо використовується експоненціальне представлення, регістр літери, що стоїть перед експонентою, визначається регістром описувача формату (буква "E" відповідає "G", "e" відповідає "g"). Експонента містить не менше двох цифр. Це відрізняє даний формат від експоненційного запису, який створюється при використанні описувача експоненціального формату, оскільки в останньому випадку експонента містить не менше трьох цифр.

Форматування результуючого рядка визначається відомостями про форматування в поточному об'єкті `NumberFormatInfo`. У таблиці 2.14

представлені властивості NumberFormatInfo, які забезпечують управління форматуванням результуючого рядка.

Таблиця 2.14 – Значення властивості NumberFormatInfo

Значення	Опис
NegativeSign	Визначає рядок, який вказує на те, що число є негативним
NumberDecimalSeparator	Визначає рядок, що розділяє цілу і дробову частини числа
PositiveSign	Визначає рядок, який вказує, що експонента є позитивною

У наступному прикладі різні значення з плаваючою комою форматируются за допомогою специфікатор загального формату.

```
double number;
number = 12345.6789;
Console::WriteLine(number.ToString("G",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 12345.6789

Console::WriteLine(number.ToString("G",
     Globalization::CultureInfo::CreateSpecificCulture("fr-FR")));
// Відобразиться 12345,6789

Console::WriteLine(number.ToString("G7",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 12345.68

number = .0000023;
Console::WriteLine(number.ToString("G",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 2.3E-06

Console::WriteLine(number.ToString("G",
     Globalization::CultureInfo::CreateSpecificCulture("fr-FR")));
// Відобразиться 2,3E-06

number = .0023;
Console::WriteLine(number.ToString("G",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 0.0023

number = 1234;
Console::WriteLine(number.ToString("G2",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 1.2E+03

number = Math::PI;
Console::WriteLine(number.ToString("G5",
```

```
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 3.1416
```

2.6.6 Описувач числового формату ("N")

Специфікатор числового формату ("N") перетворює число в рядок виду "-d,ddd,ddd.ddd ...", де знак "-" при необхідності представляє знак негативного числа, символ "d" означає цифру (0-9), знак "," - роздільник груп, а знак "." - роздільник цілої та дробової частини. Необхідна кількість знаків дробової частини задається специфікатором точності. Якщо описувач точності не заданий, число десяткових знаків визначається поточною властивістю NumberFormatInfo::NumberDecimalDigits.

Форматування результуючого рядка визначається відомостями про форматування в поточному об'єкті NumberFormatInfo. У таблиці 2.15 представлені властивості NumberFormatInfo, для забезпечення управління форматуванням результуючого рядка.

Таблиця 2.15 – Значення властивості NumberFormatInfo

Значення	Опис
NegativeSign	Визначає рядок, який вказує, що число є негативним
NumberNegativePattern	Визначає формат від'ємних значень і вказує, як саме подається негативний знак: круглими дужками або властивістю NegativeSign
NumberGroupSizes	Визначає число цифр цілої частини, що стоять між роздільниками груп
NumberGroupSeparator	Визначає рядок, що розділяє групи цифр цілої частини
NumberDecimalSeparator	Визначає рядок, що розділяє цілу і дробову частини числа
NumberDecimalDigits	Визначає кількість цифр дробової частини за замовчуванням. Це значення можна перевизначити за допомогою описувача точності

У наступному прикладі різні значення з плаваючою комою форматируются за допомогою специфікатор числового формату.

```
double dblValue = -12445.6789;
Console::WriteLine(dblValue.ToString("N",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться -12,445.68

Console::WriteLine(dblValue.ToString("N1",
     Globalization::CultureInfo::CreateSpecificCulture("sv-SE")));
// Відобразиться -12 445,7

int intValue = 123456789;
Console::WriteLine(intValue.ToString("N1",
```

```
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 123,456,789.0
```

2.6.7 Описувач відсоткового формату ("P")

При використанні описувача формату відсотка ("P") число множиться на 100 і перетворюється в рядок, що представляє відсоткову частку. Необхідна кількість знаків дробової частини задається специфікатором точності. Якщо описувач точності відсутній, то використовується значення точності числа за замовчуванням, задане властивістю `PercentDecimalDigits`.

У таблиці 2.16 представлені властивості `NumberFormatInfo`, щоб забезпечити управління форматуванням повернутого рядка.

Таблиця 2.16 – Значення властивості `NumberFormatInfo`

Значення	Опис
`PercentPositivePattern`	Визначає положення символу відсотка в позитивних значеннях
`PercentNegativePattern`	Визначає положення символу відсотка і негативного знака в негативних значеннях
`NegativeSign`	Визначає рядок, який вказує, що число є негативним
`PercentSymbol`	Визначає символ відсотка
`PercentDecimalDigits`	Визначає кількість цифр дробової частини в значенні відсотка за замовчуванням. Це значення можна перевизначити за допомогою описувача точності
`PercentDecimalSeparator`	Визначає рядок, що розділяє цілу і дробову частини числа
`PercentGroupSeparator`	Визначає рядок, що розділяє групи цифр цілої частини
`PercentGroupSizes`	Визначає число цілочисельних цифр, що входять в групу

У наступному прикладі значення з плаваючою комою форматуются за допомогою специфікатора відсоткового формату.

```
double number = .2468013;
     Console::WriteLine(number.ToString("P",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 24.68%

Console::WriteLine(number.ToString("P",
     Globalization::CultureInfo::CreateSpecificCulture("hr-HR")));
// Відобразиться 24,68%

Console::WriteLine(number.ToString("P1",
     Globalization::CultureInfo::InvariantCulture));
// Відобразиться 24.7%
```

2.6.8 Описувач формату зворотнього перетворення ("R")

Описувач формату зворотнього перетворення ("R") гарантує, що рядок, який отримується перетворенням з числового значення, при зворотньому перетворенні дасть те ж саме числове значення. Цей формат підтримується тільки для типів `Single`, `Double` і `BigInteger`.

Якщо за допомогою цього описувача форматується значення типу `BigInteger`, то його рядкове представлення буде містити всі значущі цифри `BigInteger`. Якщо за допомогою цього описувача форматується значення типу `Single` або `Double`, то спочатку для нього перевіряється загальний формат, при цьому для `Double` використовується 15-знакова точність, а для `Single` - 7-знакова точність. Якщо рядок може бути розібраний так, щоб змінна прийняла те ж значення, форматування проводиться за допомогою загального покажчика формату. Якщо при зворотному перетворенні значення все ж змінюється, то кількість значущих цифр збільшується до 17 для типу `Double` і 9 - для типу `Single`.

Хоча описувач точності можна вказати, він буде проігнорований. Наведені покажчики зворотнього перетворення в даному випадку мають перевагу перед покажчиком точності.

Форматування результуючого рядка визначається відомостями про форматування в поточному об'єкті `NumberFormatInfo`. У таблиці 2.17 представлені властивості `NumberFormatInfo`, для забезпечення управління форматуванням результуючого рядка.

Таблиця 2.17 – Значення властивості `NumberFormatInfo`

Значення	Опис
`NegativeSign`	Визначає рядок, який вказує на те, що число є негативним
`NumberDecimalSeparator`	Визначає рядок, що розділяє цілу і дробову частини числа
`PositiveSign`	Визначає рядок, який вказує, що експонента є позитивною

У наступному прикладі значення `Double` форматируются за допомогою специфікатор формату зворотнього перетворення.

```
double dbValue;
dbValue = Math::PI;
Console::WriteLine(dbValue.ToString("r"));
// Відобразиться 3.1415926535897931

Console::WriteLine(dbValue.ToString("r",
     Globalization::CultureInfo::CreateSpecificCulture("fr-FR")));
// Відобразиться 3,1415926535897931

dbValue = 1.623e-21;
```

```
Console::WriteLine(dbValue.ToString("r"));
      // Відобразиться 1.623E-21
```

2.6.9 Описувач шістнадцятиричного формату ("X")

При використанні описувача шістнадцятирічного формату ("X") число перетворюється в рядок шістнадцятирічних цифр. Регістр шістнадцятирічних цифр більше 9 збігається з регістром описувачу формату. Наприклад, щоб отримати запис "ABCDEF", задайте описувач "X" або, навпаки, задайте описувач "x", щоб отримати "abcdef". Цей формат доступний тільки для цілих типів.

Мінімальна кількість знаків у вихідному рядку задається специфікатором точності. Відсутні знаки в рядку замінюються нулями.

Форматування результуючого рядка не залежить від відомостей про форматування в поточному об'єкті NumberFormatInfo.

У наступному прикладі значення Int32 форматируются за допомогою специфікатор шістнадцятирічного формату.

```
int hexValue;
hexValue = 0x2045e;
Console::WriteLine(hexValue.ToString("x"));
// Відобразиться 2045e

Console::WriteLine(hexValue.ToString("X"));
// Відобразиться 2045E

Console::WriteLine(hexValue.ToString("X8"));
// Відобразиться 0002045E

hexValue = 123456789;
Console::WriteLine(hexValue.ToString("X"));
// Відобразиться 75BCD15

Console::WriteLine(hexValue.ToString("X2"));
// Відобразиться 75BCD15
```

2.7 Перетворення рядкових значень в числові методом Parse()

Функція Parse() перетворює рядкове представлення числа в еквівалентне йому число.

Перелік перевантажених версій функції Parse() наведений в табл. 2.18.

Таблиця 2.18 – Перевантажені версії функції Parse()

Ім'я	Опис
Parse(String^)	Перетворює рядкове представлення числа в еквівалентне йому число
Parse(String^, IFormatProvider^)	Перетворює рядкове представлення числа, записане в форматі, відповідному певній мові і регіональним параметрам, в еквівалентне йому число
Parse(String^, NumberStyles)	Перетворює рядкове представлення числа в зазначеному стилі в еквівалентне йому число
Parse(String^, NumberStyles, IFormatProvider^)	Перетворює рядкове представлення числа в зазначеному стилі і з використанням формату, відповідного даній мові і регіональним параметрам, в еквівалентну йому число

Існують перевантажені версії функції для різних числових типів. Наприклад, для типу float метод Parse матиме оголошення наступного виду:

```
public:
      static float Parse    (String^ s,
                             NumberStyles style,
                             IFormatProvider^ provider)
```

Поверненим значенням внаслідок роботи даної функції буде число з плаваючою комою одиночної точності, еквівалентне числовим значенням або символу, вказаному в параметрі s.

Опис параметрів даної перевантаженої функції наведений в табл. 2.19.

Таблиця 2.19 – Опис параметрів методу Single::Parse (String^, NumberStyles, IFormatProvider^)

Параметр	*Тип параметру*	Опис
s	System::String^	Рядок, який містить число для перетворення
style	System::Globalization ::NumberStyles	Побітовое поєднання значень перерахування, що позначають елементи стилю, які можуть бути представлені в параметрі s. Зазвичай вказується значення Float в поєднанні зі значенням AllowThousands
provider	System::IFormatProvider^	Об'єкт, який надає відомості про форматування параметра s в залежності від мови та регіональних параметрів

Параметр `style` являє собою перерахування `NumberStyles`, яке визначає стилі цілочисельних числових типів та типів з плаваючою комою, що передаються методам `Parse` та `TryParse`. Параметр `style` може містити один чи кілька прапорів з перерахування `NumberStyles`. Елементи, які може містити перерахування `NumberStyles` наведені в таблиці 2.20.

Таблиця 2.20 – Прапори перерахування `NumberStyles`

Ім’я	Опис
AllowCurrencySymbol	Вказує, що числовий рядок може містити символ валюти
AllowDecimalPoint	Вказує, що числовий рядок може включати десятковий роздільник
AllowExponent	Вказує, що числовий рядок може бути в експоненційному представленні
AllowHexSpecifier	Вказує, що числовий рядок представляє шістнадцяткове значення
AllowLeadingSign	Вказує, що числовий рядок може мати знак на початку
AllowLeadingWhite	Вказує, що початкові пробільні символи можуть бути представлені в рядку
AllowParentheses	Вказує, що числовий рядок може мати одну пару дужок, в яку укладено число. Круглі дужки вказують, що аналізований рядок є від'ємним числом
AllowThousands	Вказує, що числовий рядок може мати роздільники груп, такі як символи, що відокремлюють сотні від тисяч
AllowTrailingSign	Вказує, що числовий рядок може включати кінцевий знак
AllowTrailingWhite	Вказує, що кінцеві знаки-роздільники можуть бути представлені в аналізованому рядку
Any	Вказує, що всі стилі, за винятком AllowHexSpecifier використовуються. Це стилі складеного числа
Currency	Вказує, що всі стилі, крім AllowExponent і AllowHexSpecifier використовуються. Це стилі складеного числа
Float	Вказує, що стилі AllowLeadingWhite, AllowTrailingWhite, AllowLeadingSign, AllowDecimalPoint, і AllowExponent використовуються. Це стилі складеного числа
HexNumber	Вказує, що стилі AllowLeadingWhite, AllowTrailingWhite, і AllowHexSpecifier використовуються. Це стилі складеного числа
Integer	Вказує, що стилі AllowLeadingWhite, AllowTrailingWhite, і AllowLeadingSign використовуються. Це стилі складеного числа
None	Вказує, що немає елементів стилю, таких як початкові або кінцеві пробіли, роздільники тисяч або десятковий розділовий знак, які можуть бути представлені в рядку розбору. Аналізований рядок повинен містити тільки цілі десяткові числа
Number	Вказує, що стилі AllowLeadingWhite, AllowTrailingWhite, AllowLeadingSign, AllowTrailingSign, AllowDecimalPoint, і AllowThousands використовуються. Це стилі складеного числа

Приклади застосування функції з різними прапорами даних наведені в лістингу 2.11.

Лістинг 2.11

```
using namespace System;
using namespace System::Text;
using namespace System::Globalization;

int main()
{
      // Обробити рядок, як шістнадцяткове значення і відображати
      // значення у десятковому вигляді
      String^ numberString = "A";
      int stringValue = Int32::Parse(numberString,
            NumberStyles::HexNumber);
      Console::WriteLine("Шістнадцятиричне {0} в десятковому
            вигляді буде {1}", numberString, stringValue);

      // Розбираємо рядок, зі збереженням початкового знаку, і ігноруємо
      // початкові і кінцеві пробіли
      numberString = "    -45   ";
      stringValue = Int32::Parse(numberString, NumberStyles
            ::AllowLeadingSign | NumberStyles::AllowLeadingWhite |
                  NumberStyles::AllowTrailingWhite);
      Console::WriteLine("'{0}' перетворене на int буде '{1}'.",
                  numberString, stringValue);

      // Розбираємо рядок, з круглими дужками та міняємо їх на мінус
      // перед числом, і ігноруємо початкові і кінцеві пробіли
      numberString = "    (37)   ";
      stringValue = Int32::Parse(numberString, NumberStyles
            ::AllowParentheses | NumberStyles::AllowLeadingSign |
            NumberStyles::AllowLeadingWhite |
            NumberStyles::AllowTrailingWhite);

      Console::WriteLine("'{0}' перетворене на int буде '{1}'.",
            numberString, stringValue);
}
```

Результатом роботи модуля буде наступний текст в консолі:

```
Шістнадцятиричне A в десятковому вигляді буде 10
'    -45   ' перетворене на int буде '-45'.
'    (37)   ' перетворене на int буде '-37'.
```

2.8 Перевірка чи рядкове значення не пусте методом IsNullOrEmpty()

Метод String::IsNullOrEmpty(String^) вказує, чи є зазначений рядок рядком null або пустим (empty).

```
public:
	static bool IsNullOrEmpty(String^ value)
```

Дана функція повертає значення типу System::Boolean. Функція поверне значення true, якщо параметр value дорівнює null або є порожнім рядком (""); в іншому випадку поверне значення false.

IsNullOrEmpty() зручний метод, який дозволяє одночасно виконувати тестування чи String - null або має значення Empty. Це еквівалентно наступному коду:

```
result = s == nullptr || s == String::Empty;
```

Можна використовувати також метод IsNullOrWhiteSpace(), який дозволяє перевірити, чи є рядок null, його значення дорівнює String::Empty, або він містить тільки пробіли (WhiteSpace).

Рядок має значення null якщо йому не було присвоєно значення (в C++ і Visual Basic), або якщо явно присвоєно значення null (в C++ nullptr). Хоча функції складеного форматування можуть коректно обробляти порожній рядок, як показано в наступному прикладі, спроба викликати функцію рядка Length призведе до виключення NullReferenceException.

```
String^ s;
Console::WriteLine("Значення рядка '{0}'", s);
try
{
	Console::WriteLine("Довжина рядка складає {0}.", s->Length);
}
catch (NullReferenceException^ e)
{
	Console::WriteLine(e->Message);
}
```

Результатом роботи приклада буде:

```
Значення рядка ''
Object reference not set to an instance of an object.
```

Рядок порожній, якщо йому явно присвоюється порожній рядок ("") або String::Empty. Порожній рядок має довжину Length 0. У наступному прикладі створюється порожній рядок і відображається його значення і його довжина.

```
String^ s = "";
Console::WriteLine("Довжина рядка '{0}' складає {1}.", s, s->Length);
```

Результатом роботи приклада буде:

```
Довжина рядка '' складає 0.
```

У наступному прикладі розглядаються три рядки і визначається чи кожен рядок має значення, є пустим рядком або має значення null (лістинг 2.12).

Лістинг 2.12

```
#include "stdafx.h"

using namespace System;

String^ Test(String^ s)
{
      if (String::IsNullOrEmpty(s))
            return "має значення null або пустий";
      else
            return String::Format("(\"{0}\") не null і не пустий", s);
}

int main()
{
      String^ s1 = "abcd";
      String^ s2 = "";
      String^ s3 = nullptr;
      Console::WriteLine("Рядок s1 {0}.", Test(s1));
      Console::WriteLine("Рядок s2 {0}.", Test(s2));
      Console::WriteLine("Рядок s3 {0}.", Test(s3));
}
```

Результатом роботи даного прикладу буде:

```
Рядок s1 ("abcd") не null і не пустий.
Рядок s2 має значення null або пустий.
Рядок s3 має значення null або пустий.
```

2.9 Створення діалогових вікон за допомогою класу MessageBox

Клас MessageBox призначений для відображення діалогових вікон з повідомленнями.

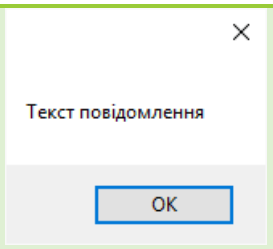
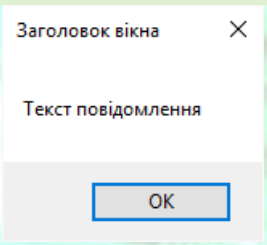
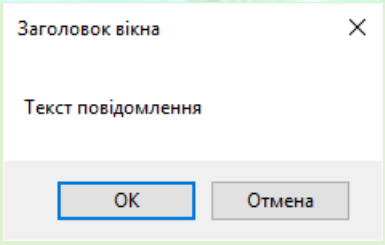
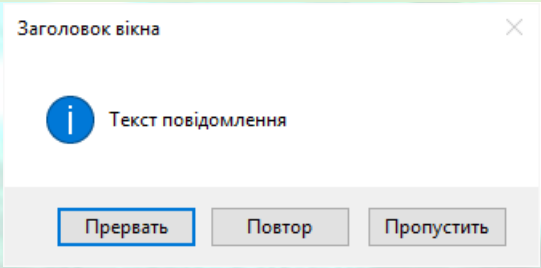
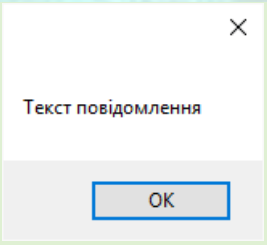
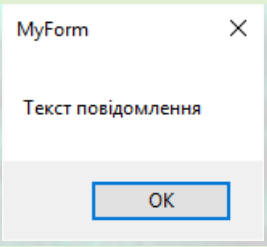
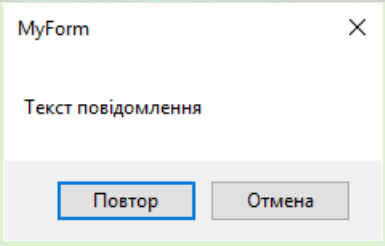
За відображення діалогового вікна відповідає метод Show() класу MessageBox. В табл. 2.21 наведений перелік перевантажених методів Show().

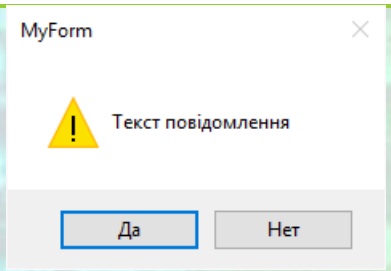
Таблиця 2.21 – Список перевантажень функції Show() класу MessageBox

№	*Ім'я*	Опис
1	*Show(String^)*	Відображає діалогове вікно з повідомленням, яке повертає результат
2	*Show(String^, String^)*	Відображає діалогове вікно з повідомленням і заголовком та повертає результат
3	*Show(String^, String^, MessageBoxButton)*	Відображає діалогове вікно з повідомленням, заголовком, кнопками і повертає результат
4	*Show(String^, String^, MessageBoxButton, MessageBoxIcon)*	Відображає діалогове вікно з повідомленням, заголовком, кнопками і значком та повертає результат
5	*Show(Window^, String^)*	Відображає вікно повідомлення перед зазначеним вікном. Діалогове вікно відображає повідомлення і повертає результат
6	*Show(Window^, String^, String^)*	Відображає вікно повідомлення перед зазначеним вікном. Діалогове вікно відображає повідомлення, заголовок та повертає результат
7	*Show(Window^, String^, String^, MessageBoxButton)*	Відображає вікно повідомлення перед зазначеним вікном. У вікні з'явиться повідомлення, рядок заголовку та кнопки і діалогове вікно поверне результат
8	*Show(Window^, String^, String^, MessageBoxButton, MessageBoxIcon)*	Відображає вікно повідомлення перед зазначеним вікном. У вікні повідомлення на екрані з'явиться повідомлення, рядок заголовку, кнопки, значок і воно повертає результат

В таблиці 2.22 наведені приклади використання перевантажених верісій функції Show() класу MessageBox.

Таблиця 2.22 – Приклади перевантажень функції Show( ) класу MessageBox

№	Ім'я	Опис
1	`MessageBox::Show("Текст повідомлення");`	
2	`MessageBox::Show("Текст повідомлення", "Заголовок вікна");`	
3	`MessageBox::Show("Текст повідомлення", "Заголовок вікна", MessageBoxButtons::OKCancel);`	
4	`MessageBox::Show("Текст повідомлення", "Заголовок вікна", MessageBoxButtons::AbortRetryIgnore, MessageBoxIcon::Information);`	
5	`MessageBox::Show(this, "Текст повідомлення");`	
6	`MessageBox::Show(this, "Текст повідомлення", this->Text);`	
7	`MessageBox::Show(this, "Текст повідомлення", this->Text, MessageBoxButtons::RetryCancel);`	

№	Ім'я	Опис
8	`MessageBox::Show(this, "Текст повідомлення", this->Text, MessageBoxButtons::YesNo, MessageBoxIcon:: Exclamation);`	

Приклад використання діалогового вікна з аналізом результату, який повертає функція Show() наведений нижче.

```
if ((MessageBox::Show("Закрити вікно?", this->Text,
      MessageBoxButtons::YesNo, MessageBoxIcon::Question)) ==
      System::Windows::Forms::DialogResult::Yes) Close();
```

Внаслідок виконання цих рядків кода з'явиться діалогове вікно, яке зображене на рис. 2.4.

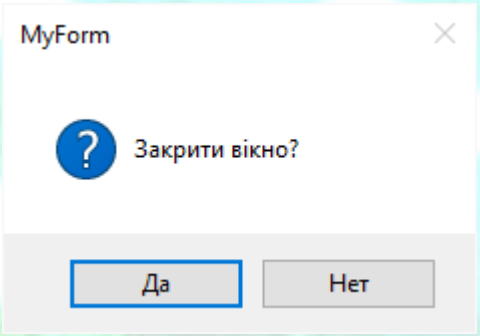


Рисунок 2.4 – Діалогове вікно з повідомленням

Якщо користувач в діалоговому вікні натисне кнопку Так, вікно закриється за допомогою метода форми `Close()`.

3 ПРИКЛАД ПРОГРАМНОЇ РЕАЛІЗАЦІЇ.

СТВОРЕННЯ ДОДАТКУ WINDOWS FORM З ЗАСТОСУВАННЯМ ВЛАСНОГО КЛАСУ

3.1 Створення додатку Windows Form

Необхідно створити настільний додаток на основі технології Microsoft .net з використанням мови CLR (Common Language Runtime) для знаходження екстремумів заданої функції з застосуванням створеного власного класу.

Для того, щоб створити проект додатка на основі технології Windows Forms необхідно виконати наступні операції.

1) Вибрати в головному меню пункт Файл ⇒ Створити ⇒ Проект… (File ⇒ New ⇒ Project…).
2) В діалоговому вікні Створення проекту (New Project) в лівій частині вікна обрати Встановлені ⇒ Шаблони ⇒ Visual C++ ⇒ CLR (Installed ⇒ Templates ⇒ Visual C++ ⇒ CLR).
3) В діалоговому вікні Створення проекту (New Project) в центральній частині вікна обрати пункт Пустий проект CLR (Empty Project CLR).
4) Задати ім'я нового проекту у полі Ім'я: (Name:).
5) Задати місце збереження нового рішення у полі Розташування: (Location:).
6) Задати ім'я нового рішення у полі Ім'я рішення: (Solution name:) та натиснути кнопку OK.

В результаті вище зазначених дій буде створений пустий проект CLR у якого відсутні будь-які елементи (рис. 3.1).

Для можливості подальшого створення віконного додатку Windows за технологією Windows Forms у створений пустий проект CLR необхідно додати пусту форму (Form), яка буде вікном майбутнього додатку.

7) Обрати в меню пункт Проект ⇒ Додати новий елемент… (Project ⇒ Add new item…).

8) З'явиться діалогове вікно Додати новий елемент (Add New Item…), в лівій частині якого слід обрати пункт Встановлені ⇒ Visual C++ ⇒ UI (Installed ⇒ Visual C++ ⇒UI).
9) Надалі в центральній частині діалогового вікна Додати новий елемент (Add New Item) обрати пункт Форма Windows Forms (Form Windows Forms).
10) У полі Ім'я: (Name:) задати ім'я нової форми (файл заголовку з розширенням .h).

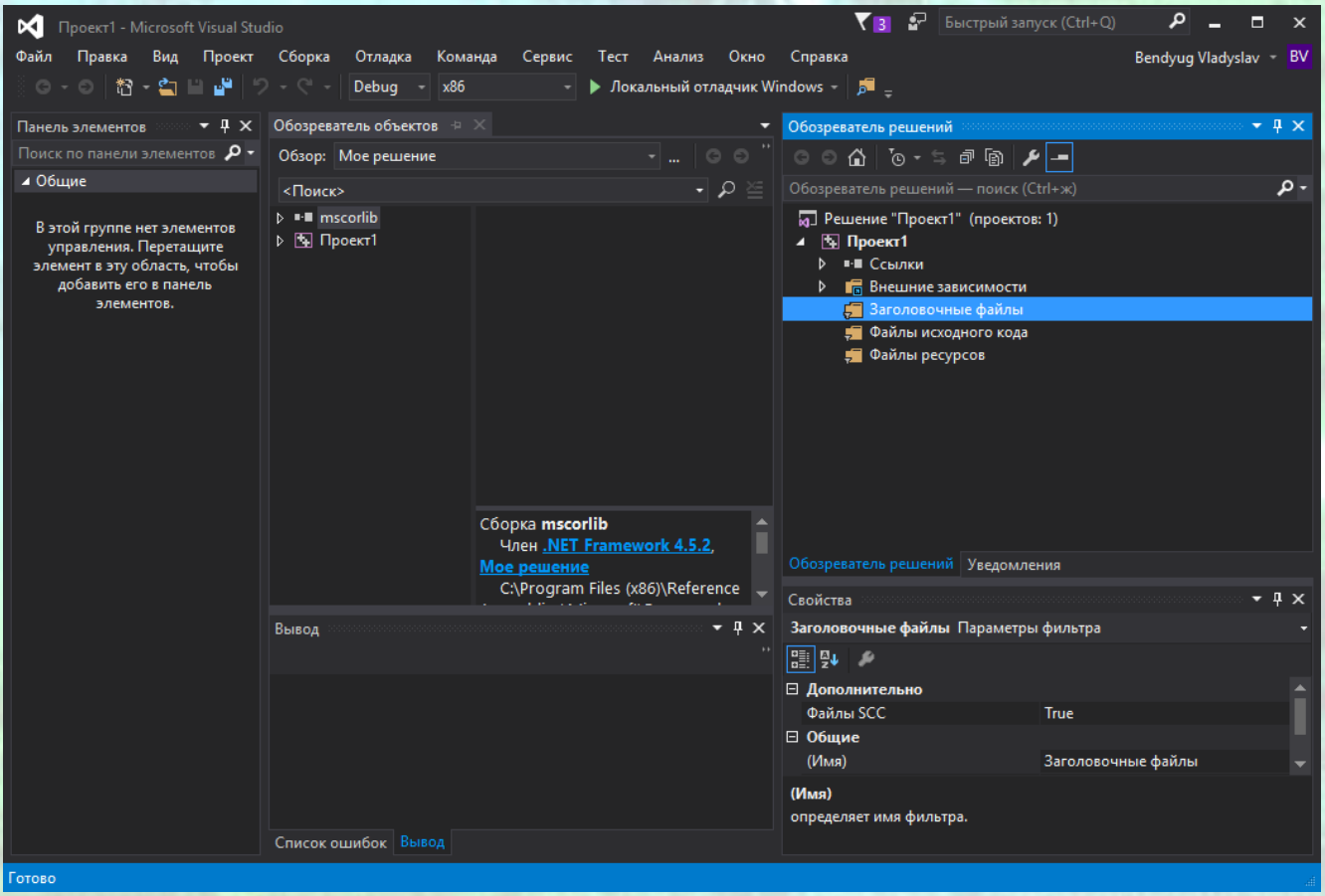


Рисунок 3.1 – Вікно середовища Visual Studio зі створеним пустим проектом CLR

11) У полі Розташування: (Location:) задається місце збереження файлу нової форми.
12) По завершенню вибору натиснути кнопку Додати (Add) для додавання нової форми у проект (рис. 3.2).
13) Надалі потрібно додати наступний програмний код у файл форми з розширенням .cpp (має таке ж ім'я, як і заголовочний файл форми), який обирається в Оглядачі рішень (Solution Explorer).

```
#include "MyForm.h"//ім'я заголовку вашої форми

using namespace Проект1; //ім'я вашого проекту

[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Application::Run(gcnew MyForm());
    return 0;
}
```

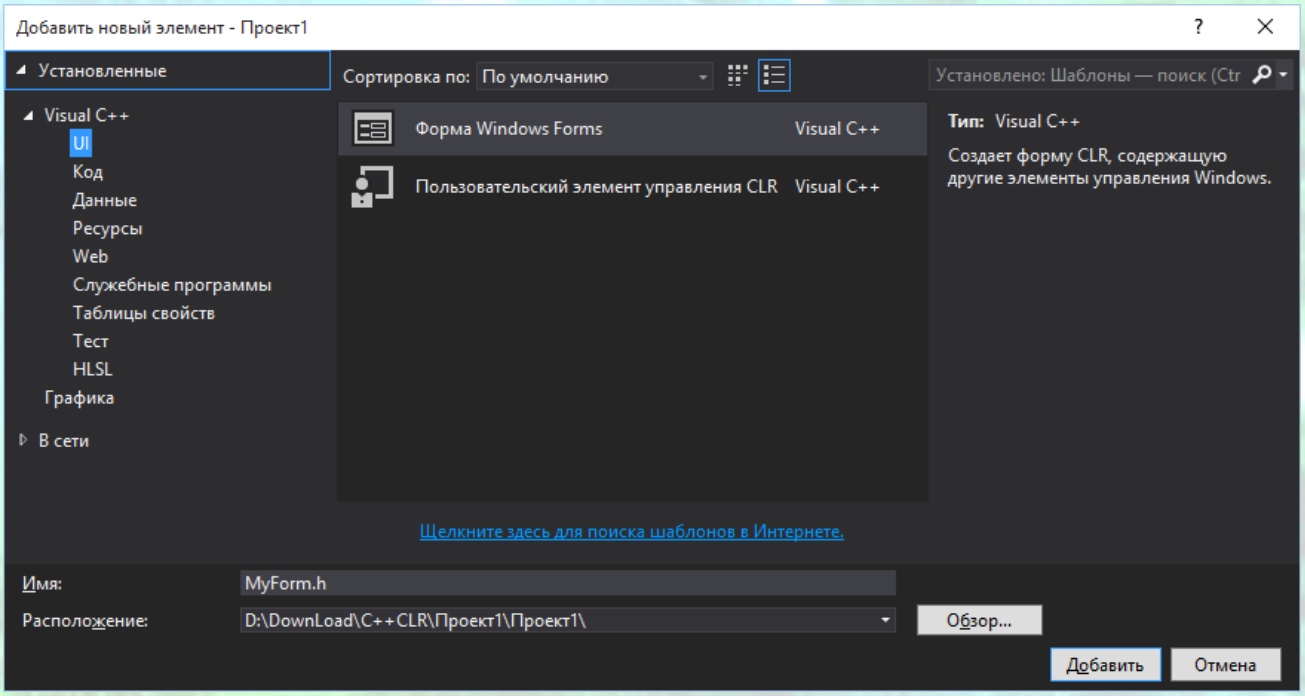


Рисунок 3.2 – Діалогове вікно додавання нового елементу до проекту CLR

14) Обрати пункт меню Проект ⇒ Властивості: Проект1 (Project ⇒ Project1 Propeties…), після чого з'явиться діалогове вікно Сторінки властивостей Проект1 (Project1 Property Pages).
15) В лівій частині діалогового вікна обираємо пункт Властивості конфігурації ⇒ Компонувальник ⇒ Система (Configuration Properties ⇒ Linker ⇒ System).
16) В правій частині діалогового вікна у полі Підсистема (SubSystem) задаємо Windows(/SUBSYSTEM:WINDOWS), як вказано на рис. 3.3.

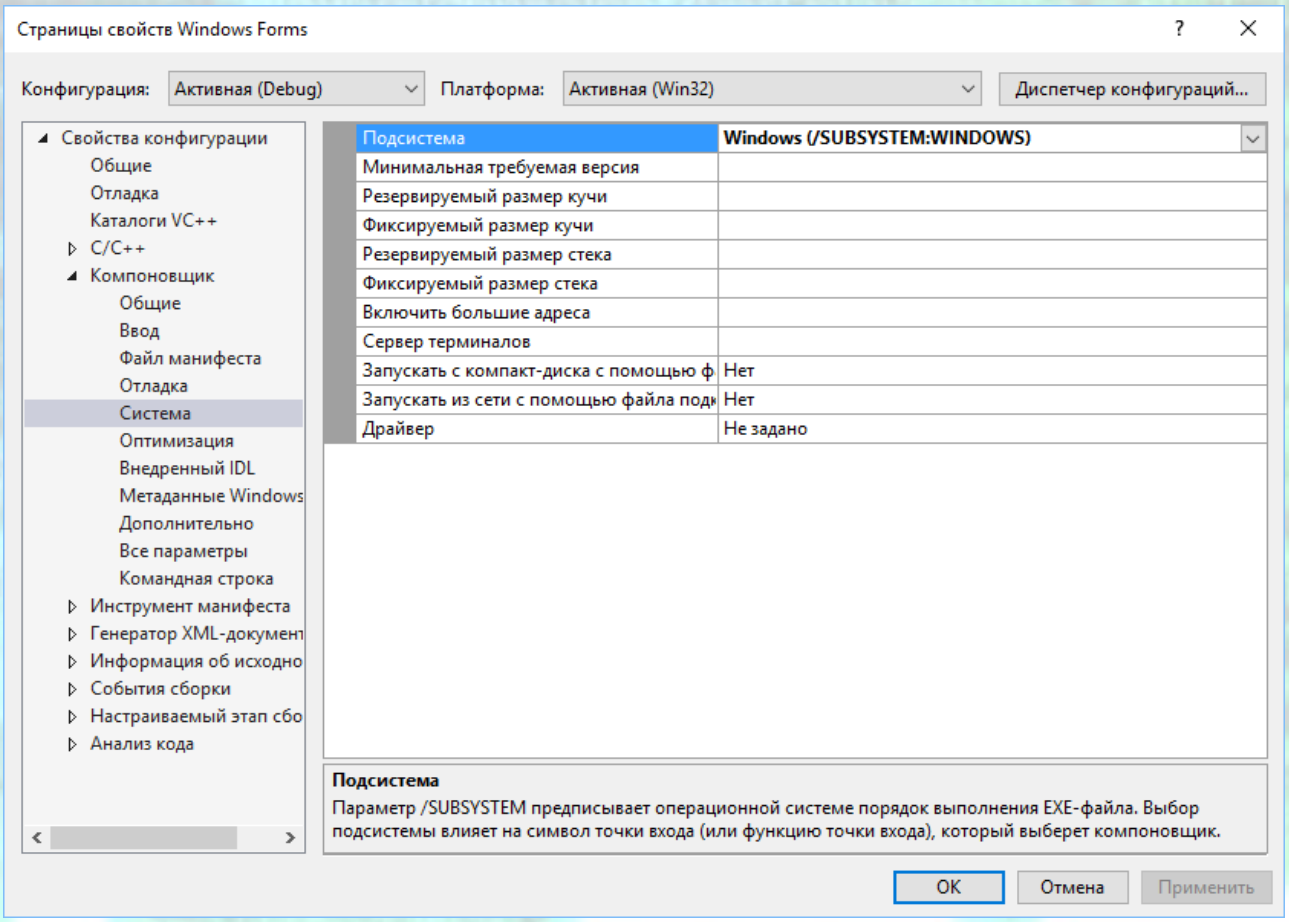


Рисунок 3.3 – Задавання значення поля Підсистема у властивостях проекту

17) В лівій частині діалогового вікна обираємо пункт Властивості конфігурації ⇒ Компонувальник ⇒ Додатково (Configuration properties ⇒ Linker ⇒ Advanced).
18) В правій частині діалогового вікна у полі Точка входу (Entry Point) задаємо main, як вказано на рис. 3.4 і натискаємо кнопку OK для застосування змін.

Після цього ми отримаємо готовий до роботи додаток Windows Forms з пустою формою у ньому. Надалі буде необхідно у вікні конструктора форми додати потрібні компоненти з вікна палітри компонентів для формування візуального інтерфейсу нашої програми.

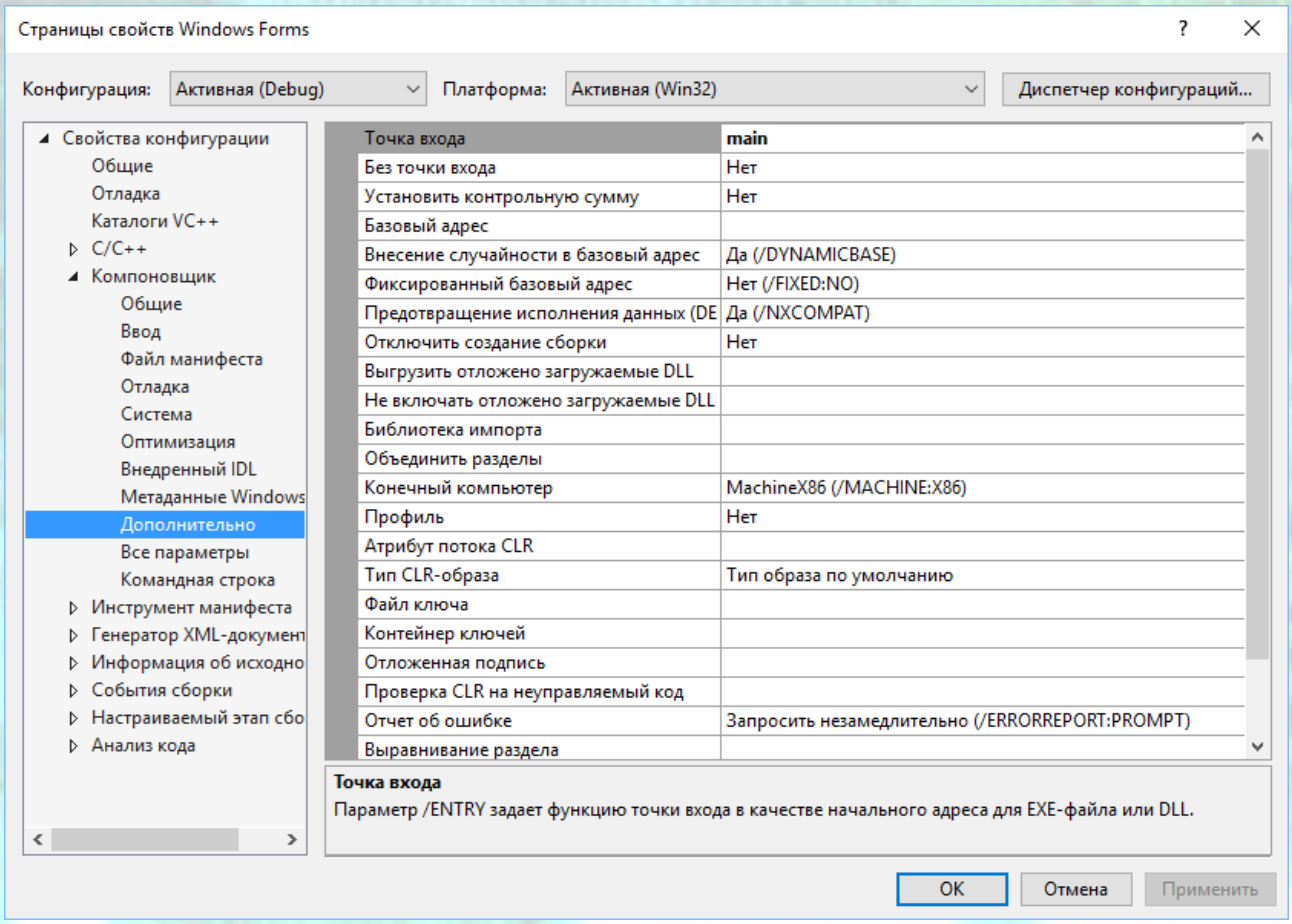


Рисунок 3.4 – Задавання значення поля Точка входу у властивостях проекту

3.2 Створення інтерфейсу програмного модуля

Наша програма повинна мати головне, контекстні меню, а також панель інструментів. Для цього з вікна Панель елементів (Toolbox), розділу Меню і панелі інструментів (Menus & Toolbars) перетягуємо у нашу форму один примірник класу MenuStrip, три примірники класу ContextMenuStrip та один примірник класу ToolStrip.

Інтерфейс програми будемо формувати у вигляді кількох сторінок з закладками. Для цього до форми потрібно додати один примірник класу TabControl з розділу Контейнери (Containers) вікна Панель елементів (Toolbox). Але для того, щоб об'єкти класів TabControl та ToolStrip не перекривали один одного після встановлення потрібних значень для властивостей Dock, необхідно спочатку додати контейнер класу Panel у форму, в якому потім і розмістимо примірник класу TabControl. Властивість Dock у Visual Studio працює некоректно і об'єкти після її зміни розташовуються не так, як очікується. Для досягнення потрібного вигляду інтерфейсу необхідно застосовувати об'єкти Panel для розміщення в них інших елементів інтерфейсу, а також використовувати властивість Anchor разом з властивістю Dock. Властивість Anchor дозволяє зафіксувати положення об'єкту відносно верхньої, нижньої, лівої та правої межі батьківського контейнеру, тобто об'єкту-контейнеру в якому розміщений даний об'єкт.

Властивості Dock об'єктам класів TabControl та Panel у вікні Властивості (Properties) встановлюємо значення Fill для того, щоб вони зайняли весь вільний простір форми (панелі). У вікні програми в нас буде чотири сторінки для розміщення елементів інтерфейсу: сторінка для введення вихідних даних, сторінка для виведення в таблицю ітерацій розрахунку, сторінка для формування текстового звіту за результатами розрахунку та сторінка для побудови графічної залежності нашої функції. Для цього в об'єкті класу TabControl створюємо чотири сторінки – примірники класу TabPage.

Створеним об'єктам надаємо індивідуальні імена для покращення читання та написання коду (табл. 3.1).

Надалі на першій сторінці tabPageДані розміщуємо об'єкти класу Panel та даємо їм імена panelЕкстремумІнтервал та panelКоефіцієнтиТочність. Панель panelЕкстремумІнтервал має бути розташована вздовж лівої межі сторінки tabPageДані, тому присвоюємо її властивості Dock значення Left. Панель panelКоефіцієнтиТочність повинна займати всю вільну область сторінки tabPageДані, тому її властивості Dock присвоюємо значення Fill.

Таблиця 3.1 – Імена об'єктів

Клас	Ім'я об'єкту
MenuStrip	menuStripГоловнеМеню
ContextMenuStrip	contextMenuStripВихідніДані
ContextMenuStrip	contextMenuStripРезультати
ContextMenuStrip	contextMenuStripГрафік
ToolStrip	toolStripПанельІнструментів
Panel	panelГоловна
TabControl	tabControlСторінки
TabPage	tabPageДані
TabPage	tabPageРезультат
TabPage	tabPageІтерації
TabPage	tabPageГрафік

В панелі panelЕкстремумІнтервал розташовуємо об'єкти groupBoxЕкстремум та groupBoxІнтервалПошуку класу GroupBox. Властивості Dock об'єкту groupBoxЕкстремум надаємо значення Top. Властивості Dock об'єкту groupBoxІнтервалПошуку надаємо значення Fill.

В панелі panelКоефіцієнтиТочність розташовуємо об'єкти groupBoxКоефіцієнти та groupBoxТочність класу GroupBox. Властивості Dock об'єкту groupBoxТочність надаємо значення Bottom. Властивості Dock об'єкту groupBoxКоефіцієнти надаємо значення Fill.

Надалі в об'єкти класу GroupBox додаємо необхідні об'єкти класів TextBox – для введення вихідних даних, Label – для текстових пояснень, та об'єкт класу CheckedListBox – для додавання опції вибору. Після цього присвоюємо необхідні властивості доданим об'єктам та розташовуємо їх належним чином, щоб перша сторінка набула наступного вигляду (рис. 3.5).

На сторінці tabPageРезультат розташовуємо об'єкт richTextBoxЗвіт класу RichTextBox для виведення в нього текстового звіту за результатами розрахунку.

На сторінці tabPageГрафік розташовуємо об'єкт pictureBox1 класу PictureBox для відкриття в ньому графічних файлів і його властивості Dock надаємо значення Fill. Також на сторінці tabPageГрафік розташовуємо об'єкт chartГрафік класу Chart для побудови в ньому графічної залежності нашої функції. Об'єкт chartГрафік також розтягуємо на всю сторінку tabPageГрафік значенням Fill властивості Dock. Але для того, щоб об'єкт chartГрафік завчасно не з'являвся у вікні програми, його властивості Visible надаємо значення False.

На сторінці tabPageІтерації розташовуємо об'єкт dataGridViewІтерації класу DataGridView для виведення у табличному

вигляді ітерацій знаходження екстремуму функції. Об'єкт `dataGridViewІтерації` також розтягуємо на всю сторінку.

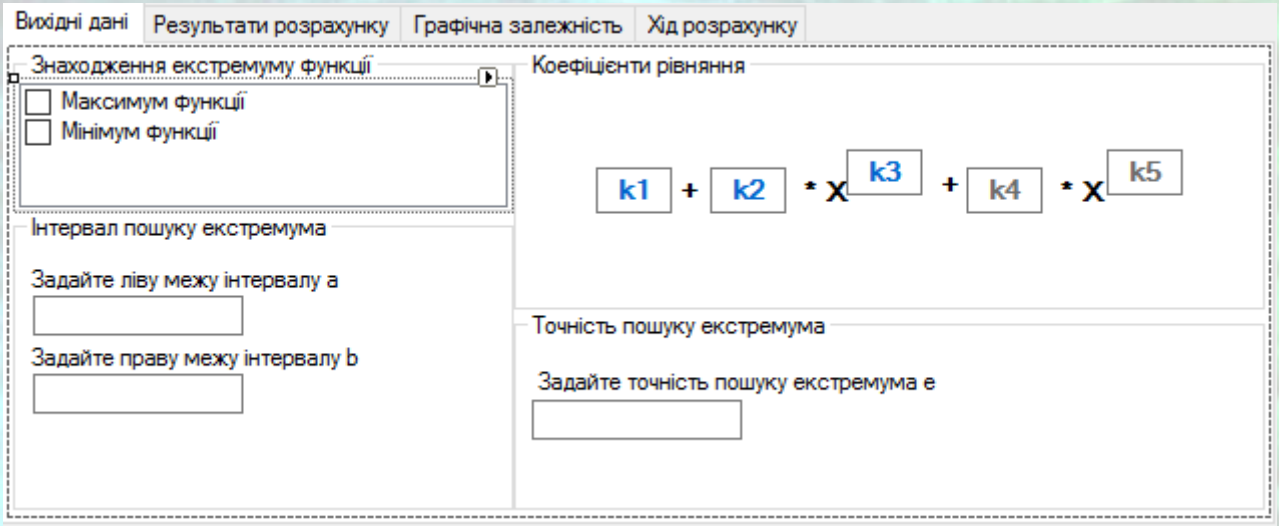


Рисунок 3.5 – Сторінка `tabPageДані` у вікні проектувальника форми

В об'єкті `toolStripПанельІнструментів` розташуємо `10` об'єктів класу `ToolStripButton` – кнопок панелі інструментів, та `5` об'єктів класу `ToolStripSeparator` – роздільників між кнопками панелі інструментів.

Задамо потрібні значення властивостям у вікні `Властивості` (`Properties`):

- у властивості `Text` об'єктів `ToolStripButton` впишемо текстове пояснення призначення кнопки панелі інструментів;
- у властивості `Image` додамо піктограмки, які будуть відображатися на кнопках панелі інструментів;
- у властивості `ToolTipText` впишемо текст спливаючих підказок, які з'являються при наведенні покажчика миші на кнопку;
- властивості `DisplayStyle` надамо значення `ImageAndText`, для того щоб на кнопках панелі інструментів відображався текст разом з графічним зображенням;
- властивості `TextImageRelation` надамо значення `ImageAboveText`, щоб зображення розташовувалось над текстом у кнопці панелі інструментів.

Після редагування значень необхідних властивостей, панель інструментів набуде вигляду як на рис. 3.6.

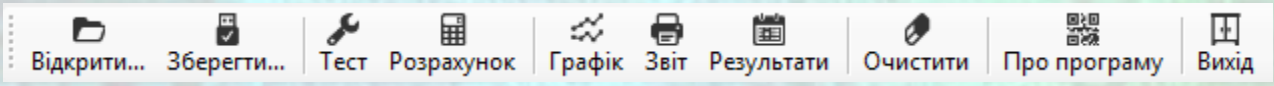


Рисунок 3.6 – Панель інструментів у вікні проектувальника форми

Створимо головне меню додатку. Для цього виділимо у редакторі форми примірник класу MenuStrip та дамо йому ім'я menuStripГоловнеМеню. За допомогою редактора меню створимо три вкладки головного меню: Файл, Розрахунок та Довідка. Додамо до вкладок меню потрібні пункти та роздільники – примірники класу ToolStripItem, та змінимо необхідні властивості як вказано у таблиці 3.2.

Таблиця 3.2 – Пункти головного меню та їх властивості

Група меню	*Властивість Name*	*Властивість Text*	Властивість ShortcutKeys
Файл	*відкритиФайлToolStripMenuItem*	*Відкрити файл*	*Ctrl+O*
	зберегтиФайлToolStripMenuItem	*Зберегти файл*	*Ctrl+S*
	вихідToolStripMenuItem	*Вихід*	*Alt+F4*
Розрахунок	*тестовийПрикладToolStripMenuItem*	*Тестовий приклад*	*Alt+T*
	розрахуватиToolStripMenuItem	*Розрахувати*	*Alt+C*
	графікToolStripMenuItem	*Графік*	*Ctrl+G*
	звітToolStripMenuItem	*Звіт*	*Ctrl+R*
	таблицяРозрахункуToolStripMenuItem	*Таблиця розрахунків*	*Ctrl+T*
	очиститиToolStripMenuItem	*Очистити*	*Alt+Del*
Довідка	*проПрограмуToolStripMenuItem*	*Про програму*	*F1*

Вкладки головного меню програми після присвоєння властивості Image відповідних графічних зображень повинні мати наступний вигляд (рис. 3.7).

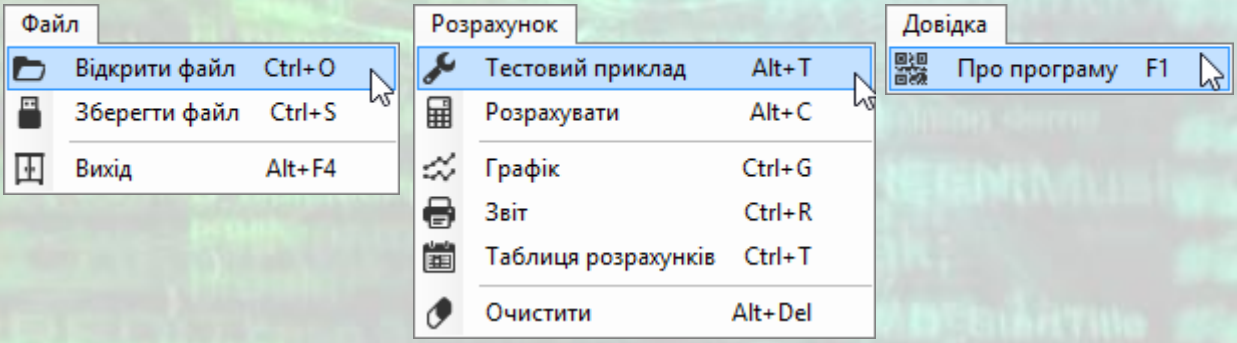


Рисунок 3.7 – Пункти головного меню програми

Надалі створимо три контекстних меню та встановимо їм відповідні властивості (табл. 3.3).

Таблиця 3.3 – Пункти головного меню та їх властивості

Об'єкт меню	*Властивість Name*	*Властивість Text*	Властивість ShortcutKeys
contextMenu-StripВихідніДані	*тестовийПриклад-ToolStripMenuItem1*	*Тестовий приклад*	*Alt+T*
	розрахуватиToolStrip-MenuItem1	*Розрахувати*	*Alt+C*
	очиститиToolStripMenuItem1	*Очистити*	*Alt+Del*
contextMenu-StripРезультати	*відкритиФайл-ToolStripMenuItem1*	*Відкрити файл RTF*	*Ctrl+O*
	зберегтиФайл-ToolStripMenuItem1	*Зберегти файл RTF*	*Ctrl+S*
	виділитиВсеToolStripMenuItem	*Виділити все*	*Ctrl+A*
	скопіюватиToolStripMenuItem	*Скопіювати*	*Ctrl+C*
	вирізатиToolStripMenuItem	*Вирізати*	*Ctrl+X*
	вставитиToolStripMenuItem	*Вставити*	*Ctrl+V*
	видалитиToolStripMenuItem	*Очистити*	*Alt+Del*
	відмінитиToolStripMenuItem	*Відмінити*	*Ctrl+Z*
contextMenu-StripГрафік	*відкритиГрафічнийФайл-ToolStripMenuItem*	*Відкрити графічний файл*	*Ctrl+O*
	зберегтиГрафічнийФайл-ToolStripMenuItem	*Зберегти графічний файл*	*Ctrl+S*

Для прив'язки контекстного меню contextMenuStripВихідніДані до сторінки введення даних Вихідні дані, встановимо для властивості ContextMenuStrip об'єкту tabPageДані значення contextMenuStripВихідніДані (рис. 3.8).

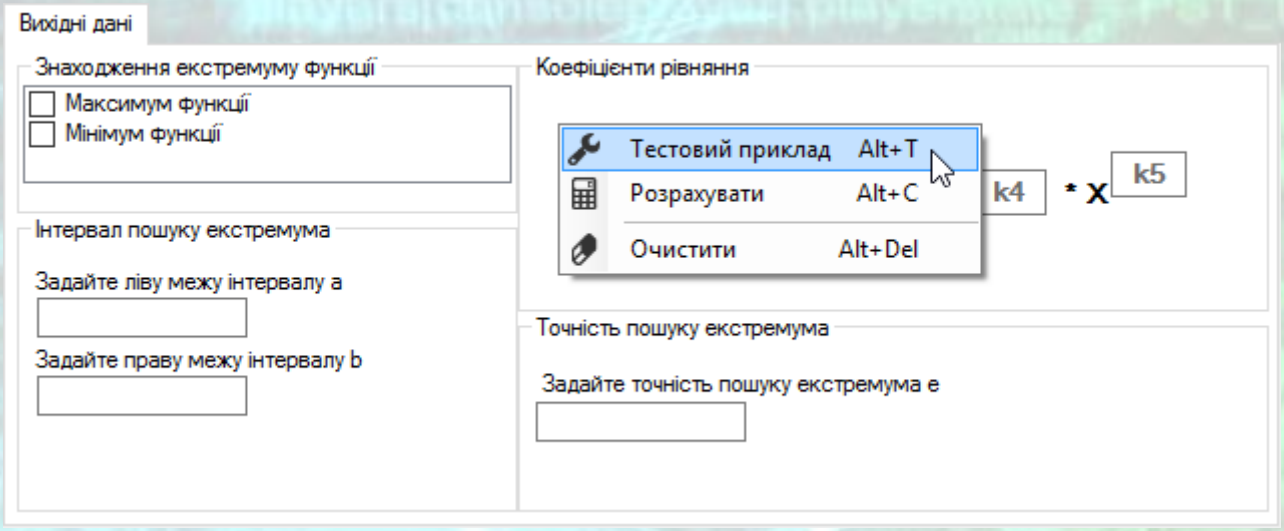


Рисунок 3.8 – Прив'язка контекстного меню до сторінки Вихідні дані

Для прив'язки контекстного меню contextMenuStripРезультати до сторінки Результати розрахунку створення звіту за результатами розрахунку, встановимо для властивості ContextMenuStrip об'єкту tabPageРезультат значення contextMenuStripРезультати (рис. 3.9).

Для прив'язки контекстного меню contextMenuStripГрафік до сторінки формування графіку функції Графічна залежність, встановимо для властивості ContextMenuStrip об'єкту tabPageГрафік значення contextMenuStripГрафік (рис. 3.10).

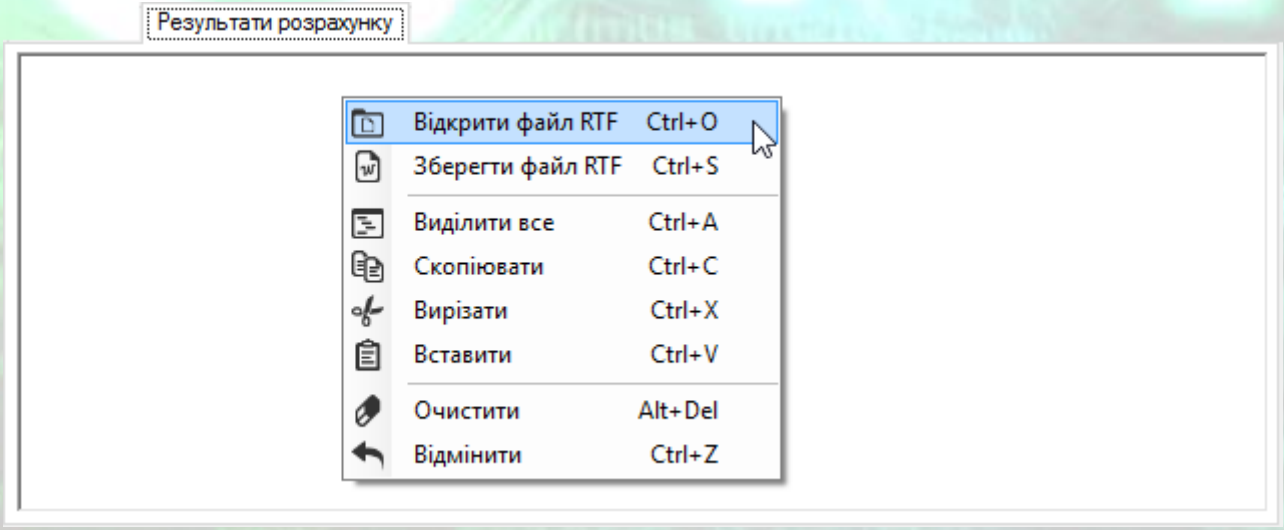


Рисунок 3.9 – Прив'язка контекстного меню до сторінки Результати розрахунку

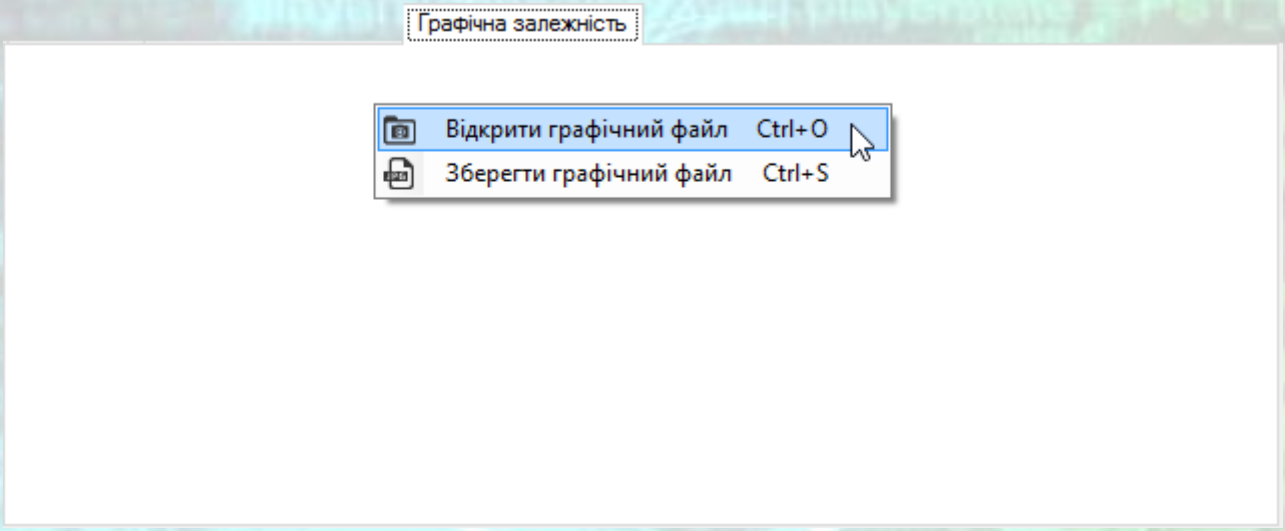


Рисунок 3.10 – Прив'язка контекстного меню до сторінки `Графічна залежність`

На цьому основний етап створення графічного інтерфейсу програмного модуля завершено.

Надалі необхідно запрограмувати реакцію на подію певних елементів інтерфейсу та запрограмувати безпосередньо потрібний алгоритм розрахунку.

3.3 Створення зовнішнього класу

3.3.1 Додавання файлу заголовку до проекту

Згідно з прикладом програмної реалізації необхідно створити програмний модуль для знаходження екстремумів заданої функції з застосуванням створеного власного класу. Екстремум функції будемо шукати з застосуванням методу золотого перетину. Даний метод необхідно реалізувати у власному зовнішньому класі. Для цього додамо до нашого проекту заголовочний файл та дамо йому ім'я `MyMethod.h`. Після додавання чи відкриття нового файлу у режимі редагування проекту потрібно включити новий файл до складу нашого проекту (рішення). Для цього потрібно щоб новий файл був відкритий і його вкладка була поточною у редакторі коду. Надалі необхідно з меню `Файл` обрати пункт `Перемістити <Ім'я файлу> в ⇒ <Ім'я рішення>`, як показано на рис. 3.11.

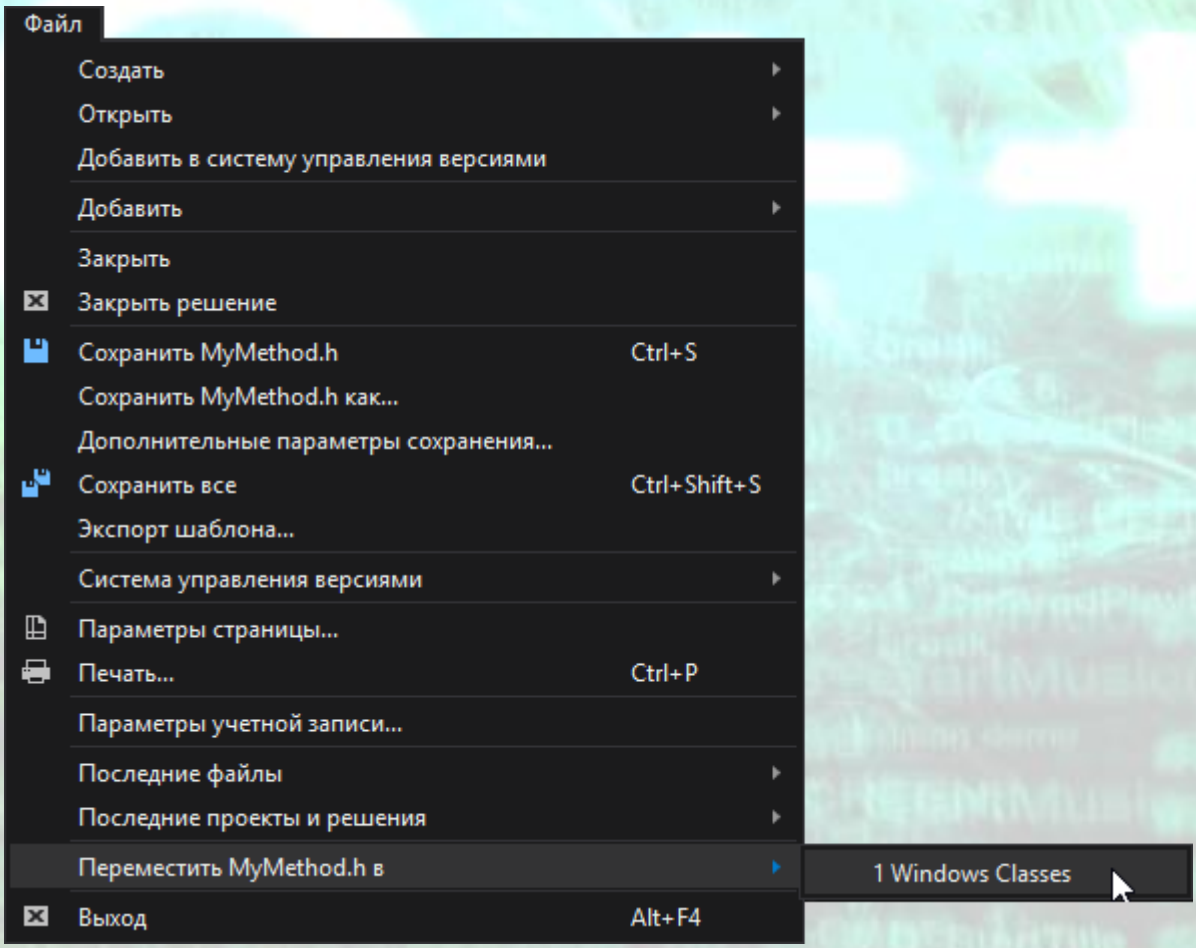


Рисунок 3.11 – Включення заголовочного файлу `MyMethod.h` до складу рішення `Windows Classes`

У разі вдалого додавання файлу до складу рішення, він повинен з'явитися у вікні Оглядача рішень (Solution Explorer), як показано на рис. 3.12.

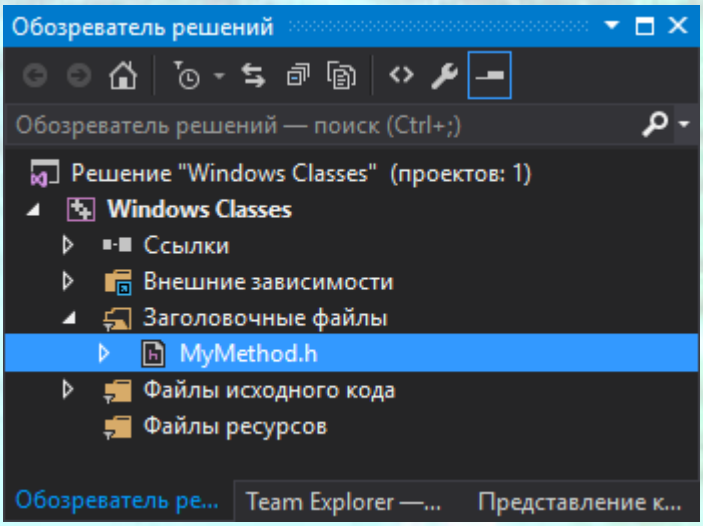


Рисунок 3.12 – Файлу заголовку MyMethod.h у складу рішення Windows Classes

Надалі перейдемо безпосередньо до написання класу.

3.3.2 Оголошення класу

Шукати екстремум будемо у функції, яка має вигляд наступного багаточлену

```
y = K1 + K2*pow(x, K3) + K4*pow(x, K5)
```

де K1, K2, K3, K4 та K5 – коефіцієнти, які буде задавати користувач у вікні програмного модуля.

Екстремум будемо шукати на заданому інтервалі з вказаною точністю.

Оголошення класу, яке міститься у файлі заголовку MyMethod.h, наведено у лістингу 3.1.

Лістинг 3.1

```
public ref class MyClass
{
public: //Інтерфейс класу
      MyClass() {};//Конструктор класу за замовчуванням
      //Конструктор класу:
      MyClass(const float &k1, const float &k2, const float &k3,
            const float &k4, const float &k5, const float &a1,
            const float &b1, const float &e1, const bool &max1) :
```

```
            A(a1), B(b1), E(e1), K1(k1), K2(k2), K3(k3), K4(k4), K5(k5),
            MINM(max1) {};
        //Конструктор класу:
        MyClass(const float &k1, const float &k2, const float &k3,
            const float &a1, const float &b1, const bool &max1) :
            A(a1), B(b1), K1(k1), K2(k2), K3(k3), MINM(max1) {};
        //Об'явлення функцій-членів класу без параметрів:
        String^ Method();
        DataTable^ ТаблицяЗначень();
        DataTable^ ТаблицяРозрахунків();
        String^ ТекстовийЗвіт();
        ~MyClass() {};//Об'явлення деструктора класу
private: //Реалізація класу
        //Ініціалізація даних-членів класу:
        double K1 = 15;       //Коефіцієнт функції
        double K2 = 5;        //Коефіцієнт функції
        double K3 = 2;        //Коефіцієнт функції
        double K4 = 0;        //Коефіцієнт функції
        double K5 = 0;        //Коефіцієнт функції
        double A = -100;      //Ліва межа пошуку екстремуму
        double B = 100;       //Права межа пошуку екстремуму
        double E = 0.001;     //Точність пошуку екстремуму
        bool MINM = true;     //Шукаємо мінімум чи максимум
        double extrX, extrY;  //Значення екстремуму функції
        //Об'явлення функції-члену класу з параметром –
        //константним посиланням:
        double MyFunc(const double);
        // Об'явлення та створення об'єкта "ТаблицяІтерації"
        //класу DataTable:
        DataTable^ ТаблицяІтерації = gcnew DataTable();
        // Об'явлення та створення об'єкта "ТаблицяГрафік" класу DataTable:
        DataTable^ ТаблицяГрафік = gcnew DataTable();
        // Об'явлення та створення об'єкта "НабірДаних" класу DataSet:
        DataSet^ НабірДаних = gcnew DataSet();
        // Створення об'єкту для збереження текстового звіту:
        String^ Звіт;
};
```

3.3.3 Створення конструктора класу

Оскільки клас має створений нами конструктор, то до інтерфейсу класу необхідно додати конструктор за замовчуванням

```
MyClass() {};//Конструктор класу за замовчуванням
```

Це конструктор без параметрів, який передбачає ініціалізацію змінних-членів класу значеннями за замовчуванням. Визначення та ініціалізація змінних-членів класу розміщена у блоці реалізації класу (розділ `private:`)

```
//Ініціалізація даних-членів класу:
double K1 = 15;
double K2 = 5;
double K3 = 2;
double K4 = 0;
double K5 = 0;
double A = -100;
double B = 100;
double E = 0.001;
bool MINM = true;
double extrX, extrY;
```

У складі інтерфейсу класу розміщений конструктор, що передбачає створення примірника класу з ініціалізацією змінних-членів переданими до класу даними

```
//Конструктор класу:
MyClass(const float &k1, const float &k2, const float &k3,
      const float &k4, const float &k5, const float &a1, const float &b1,
      const float &e1, const bool &max1) : A(a1), B(b1), E(e1), K1(k1),
      K2(k2), K3(k3), K4(k4), K5(k5), MINM(max1) {};
```

В цьому конструкторі при створенні примірника класу передаються значення для ініціалізації наступних змінних-членів: A та B – межі інтервалу пошуку екстремуму; E – точність пошуку екстремуму; K1, K2, K3, K4, K5 – коефіцієнти багаточлену; MINM – логічна змінна, яка відповідає за пошук мінімуму чи максимуму функції.

Також клас має третій конструктор, який отримує дані для ініціалізації змінних-членів A та B, K1, K2, K3 та MINM

```
//Конструктор класу:
MyClass(const float &k1, const float &k2, const float &k3,
      const float &a1, const float &b1, const bool &max1) :
      A(a1), B(b1), K1(k1), K2(k2), K3(k3), MINM(max1) {};
```

Всі інші змінні-члени класу при цьому будуть ініціалізовані значеннями, які містяться у блоці реалізації класу, чи значеннями за замовчуванням.

Використання того чи іншого конструктора для створення примірника нашого класу буде залежати від кількості та типу параметрів, які будуть використовуватись при створенні змінної класу у програмі. Наприклад,

```
Екстремум = gcnew MyClass(k1, k2, k3, a, b, mmaxx);
```

де Екстремум – ім'я змінної типу класу MyClass, яка створена за допомогою оператора gcnew, тобто в динамічній купі пам'яті. При цьому коефіцієнти K4 та K5 будуть ініціалізовані значенням 0, а змінна-член E - значенням 0.001.

3.3.4 Створення функцій-членів класу

В оголошенні класу також присутні функції-члени класу, які будуть доступні користувачам класу, тобто до них можна буде звертатися в тексті програми через змінну класу. Для цього ці функцію об'явлені в інтерфейсі класу (розділ public:)

```
//Об'явлення функцій-членів класу без параметрів:
String^ Method();
DataTable^ ТаблицяЗначень();
DataTable^ ТаблицяРозрахунків();
String^ ТекстовийЗвіт();
```

Також у закритому розділі оголошення класу знаходиться оголошення функції

```
double MyFunc(const double);
```

Оголошення даної функції розташоване в розділі private:, тому вона не буде доступна за межами класу.

3.3.5 Створення функції-члену з розрахунковою залежністю

У функції MyFunc безпосередньо міститься наша залежність для якої ми шукатимемо екстремум:

Лістинг 3.2

```
//Визначення функції-члену класу з параметром - константним посиланням:
double MyClass::MyFunc(const double x)
{
	double y;
	y = K1 + K2*pow(x, K3) + K4*pow(x, K5);
	return y;
}
```

Визначення функцій-членів класу знаходиться безпосередньо після оголошення класу у файлі заголовку.

3.3.6 Створення функції-члену для знаходження екстремума

Алгоритм розрахунку – пошук екстремуму методом золотого перетину, запрограмований у функції-члені `Method()`, що наведена у лістингу 3.3.

Лістинг 3.3

```
String^ MyClass::Method() //Визначення функції-члену класу без параметрів
{
	double y1, y2, x1, x2;
	const double fi = (1 + sqrt(5)) / 2; //Золотий перетин
	unsigned i = 0;
	//Ініціалізація змінних a та b значеннями змінних A та B відповідно
	double a(A), b(B);
	String^ minmax;
	String^ Результат;
	MINM ? minmax = "мінімум" : minmax = "максимум";
	DataRow^ Ряд = ТаблицяІтерації->NewRow();
	//Додаємо в таблицю стовпчик з іменем "Ітерація"
	//та вказуємо тип його значень:
	ТаблицяІтерації->Columns->Add("Ітерація", int::typeid);
	//Додаємо в таблицю стовпчик з іменем "Значення x1"
	//та вказуємо тип його значень:
	ТаблицяІтерації->Columns->Add("Значення x1", double::typeid);
	//Додаємо в таблицю стовпчик з іменем "Значення функції y(x1)"
	//та вказуємо тип його значень:
	ТаблицяІтерації->Columns->Add("Значення функції y(x1)",
		double::typeid);
	//Додаємо в таблицю стовпчик з іменем "Значення x2"
	//та вказуємо тип його значень:
	ТаблицяІтерації->Columns->Add("Значення x2", double::typeid);
```

```
//Додаємо в таблицю стовпчик з іменем "Значення функції y(x2)"
//та вказуємо тип його значень:
ТаблицяІтерації->Columns->Add("Значення функції y(x2)",
      double::typeid);
//Задаємо ім'я таблиці даних:
ТаблицяІтерації->TableName = "Знаходження " + minmax +
      "у функції методом золотого перетину";
//Задаємо заголовок звіту:
Звіт += "Знаходження " + minmax +
     "у функції методом золотого перетину\n";
x2 = a + (b - a) / fi;
x1 = b - (b - a) / fi;
y1 = MyFunc(x1);
y2 = MyFunc(x2);
do
{
     y1 = MyFunc(x1);
     y2 = MyFunc(x2);
     //Перевіряємо що шукаємо мін чи макс:
     if ((!MINM && y1 <= y2) || (MINM && y1 >= y2))
     {
          a = x1;
          x1 = x2;
          x2 = a + (b - a) / fi;
     }
     else
     {
          b = x2;
          x2 = x1;
          x1 = b - (b - a) / fi;
     }
     i++;
     //Формуємо рядок звіту з форматованим виведенням
     //значень функцією ToString("F3"):
     Звіт += "Ітерація " + Convert::ToString(i) + ",
           x1=" + x1.ToString("F3") + ", y(x1)=" +
          y1.ToString("F3") + ", x2=" + x2.ToString("F3") + ",
           y(x2)=" + y2.ToString("F3") + "\n";
     //Формуємо рядок таблиці з округленням значень:
     Ряд["Ітерація"] = i;
     Ряд["Значення x1"] = Math::Round(x1, 3,
          ::MidpointRounding::AwayFromZero);
     Ряд["Значення функції y(x1)"] = Math::Round(y1, 3,
```

```
                ::MidpointRounding::AwayFromZero);
            Ряд["Значення x2"] = Math::Round(x2, 3,
                ::MidpointRounding::AwayFromZero);
            Ряд["Значення функції y(x2)"] = Math::Round(y2, 3,
                ::MidpointRounding::AwayFromZero);
            //Додаємо сформований рядок до таблиці:
            ТаблицяІтерації->Rows->Add(Ряд);
            //Створюємо наступний пустий рядок в таблиці:
            Ряд = ТаблицяІтерації->NewRow();
        } while (abs(b - a) >= E); //Перевірка досягнення точності
        extrX = (a + b) / 2; //Значення x в точці екстремума
        extrY = MyFunc(extrX); //Значення функції в точці екстремума
        Результат = "Знайдений " + minmax + " функції при x=" +
            extrX.ToString("F3") + " y(x)=" + extrY.ToString("F3");
        Звіт += Результат;
        // Додати об'єкт ТаблицяІтерації в DataSet
        НаборДаних->Tables->Add(ТаблицяІтерації);
        return Результат;
}
```

В даній функції-члені виконуються розрахунки зі знаходження екстремуму функції, а також проміжні розрахунки записуються у таблицю даних та формується текстовий звіт за результатами розрахунку.

Значення, яке повертає функція має тип `String^`. За результатами своєї роботи функція-член у місце свого виклику повертає рядок зі знайденим значенням екстремуму. Для цього в середині функції визначена змінна `Результат`, якій і присвоюється рядок з результатом розрахунку.

```
String^ Результат;
Результат = "Знайдений " + minmax + " функції при x=" +
    extrX.ToString("F3") + " y(x)=" + extrY.ToString("F3");
return Результат;
```

Інші результати з функції-члена передаються в клас за рахунок використання глобальних змінних-членів, які оголошені в закритій частині оголошення класу.

```
private: //Реалізація класу
    //Ініціалізація даних-членів класу:
    double extrX, extrY;  //Значення екстремуму функції
    // Об'явлення та створення об'єкта "ТаблицяІтерації"
```

```
//класу DataTable:
DataTable^ ТаблицяІтерації = gcnew DataTable();
// Об'явлення та створення об'єкта "НабірДаних" класу DataSet:
DataSet^ НабірДаних = gcnew DataSet();
// Створення об'єкту для збереження текстового звіту:
String^ Звіт;
```

В змінні `extrX` та `extrY` записується значення знайденого екстремуму.

Змінна `ТаблицяІтерації` являє собою таблицю значень. Дана таблиця формується у функції-члені `MyClass::Method()`.

```
DataRow^ Ряд = ТаблицяІтерації->NewRow();
//Додаємо в таблицю стовпчик з іменем "Ітерація"
//та вказуємо тип його значень:
ТаблицяІтерації->Columns->Add("Ітерація", int::typeid);
//Додаємо в таблицю стовпчик з іменем "Значення x1"
//та вказуємо тип його значень:
ТаблицяІтерації->Columns->Add("Значення x1", double::typeid);
//Додаємо в таблицю стовпчик з іменем "Значення функції y(x1)"
//та вказуємо тип його значень:
ТаблицяІтерації->Columns->Add("Значення функції y(x1)", double::typeid);
//Додаємо в таблицю стовпчик з іменем "Значення x2"
//та вказуємо тип його значень:
ТаблицяІтерації->Columns->Add("Значення x2", double::typeid);
//Додаємо в таблицю стовпчик з іменем "Значення функції y(x2)"
//та вказуємо тип його значень:
ТаблицяІтерації->Columns->Add("Значення функції y(x2)", double::typeid);
//Задаємо ім'я таблиці даних:
ТаблицяІтерації->TableName = "Знаходження " + minmax +
      "у функції методом золотого перетину";
```

Для цього спочатку створюється локальна змінна `Ряд` в межах функції `MyClass::Method()`, яка являє собою рядок таблиці, та додається новий рядок до пустої таблиці даних `ТаблицяІтерації` за допомогою методу `NewRow()`:

```
DataRow^ Ряд = ТаблицяІтерації->NewRow();
```

Після цього до таблиці `ТаблицяІтерації` додаються всі необхідні стовпчики – поля даних таблиці з вказуванням імені поля даних та типу його значень. Наприклад, перший стовпчик даних матиме ім'я `Ітерація` та зберігатиме цілочисельні значення – номер поточної ітерації розрахунку:

```
ТаблицяІтерації->Columns->Add("Ітерація", int::typeid);
```

Також таблиці ТаблицяІтерації присвоюється ім'я за допомогою властивості TableName класу DataTable^:

```
ТаблицяІтерації->TableName = "Знаходження " + minmax +
      "у функції методом золотого перетину";
```

Надалі у розрахунковому циклі do while полям змінної Ряд присвоюються розраховані на певній ітерації значення змінних x1, y1, x2, y2, а також порядковий номер поточної ітерації i:

```
Ряд["Ітерація"] = i;
Ряд["Значення x1"] = Math::Round(x1, 3,
     ::MidpointRounding::AwayFromZero);
Ряд["Значення функції y(x1)"] = Math::Round(y1, 3,
     ::MidpointRounding::AwayFromZero);
Ряд["Значення x2"] = Math::Round(x2, 3,
     ::MidpointRounding::AwayFromZero);
Ряд["Значення функції y(x2)"] = Math::Round(y2, 3,
     ::MidpointRounding::AwayFromZero);
```

Отримані розрахункові значення округлюємо до 3 знаків після коми за допомогою функції Round з простору імен Math:

```
Math::Round(x1, 3, ::MidpointRounding::AwayFromZero)
```

Заповнений значеннями рядок даних Ряд додається до таблиці ТаблицяІтерації та створюється новий пустий рядок таблиці:

```
//Додаємо сформований рядок до таблиці
ТаблицяІтерації->Rows->Add(Ряд);
//Створюємо наступний пустий рядок в таблиці
Ряд = ТаблицяІтерації->NewRow();
```

Після наповнення таблиці, вона додається до набору даних класу DataSet:

```
// Додати об'єкт ТаблицяІтерації в DataSet
НаборДаних->Tables->Add(ТаблицяІтерації);
```

У функції-члені MyClass::Method() також формується текстовий звіт за результатами розрахунку. Для цього використовується глобальна змінна-член Звіт з закритого розділу оголошення класу private:

```
private: //Реалізація класу
     // Створення об'єкту для збереження текстового звіту:
     String^ Звіт;
```

Розраховані на кожній ітерації значення записуються в змінну-член Звіт наступним чином:

```
//Формуємо рядок звіту з форматованим виведенням значень
//функцією ToString("F3"):
Звіт += "Ітерація " + Convert::ToString(i) + ", x1=" + x1.ToString("F3")
+ ", y(x1)=" + y1.ToString("F3") + ", x2=" + x2.ToString("F3") + ",
y(x2)=" + y2.ToString("F3") + "\n";
```

При цьому числові значення потрібно трансформувати в рядкові, оскільки змінна Звіт може містити лише текстові дані. Для цього можна скористатися методом Convert::ToString(). Даний метод перетворює значення задане у вигляді цілого числа зі знаком в еквівалентне рядкове представлення:

```
Convert::ToString(i)
```

Значення з плаваючою комою потрібно виводити з заданою точністю, тобто окрім перетворення значення в рядкове, необхідно також виконувати форматування значення. Для цього використано метод Double.ToString (String).

```
x1.ToString("F3")
```

3.3.7 Створення функції-члену для побудови графічної залежності

Для побудови графічної залежності досліджуваної функції потрібно створити таблицю залежності значення функції від її аргумента. З цією метою до складу класу включена функція-член MyClass::ТаблицяЗначень(), яка наведена в лістингу 3.4.

Лістинг 3.4

```
DataTable^ MyClass::ТаблицяЗначень()
{
	DataRow^ Ряд = ТаблицяГрафік->NewRow();
	double x(A), y;
	//Додаємо в таблицю стовпчик з іменем "Значення x"
	//та вказуємо тип його значень:
	ТаблицяГрафік->Columns->Add("Значення x", double::typeid);
	//Додаємо в таблицю стовпчик з іменем "Значення функції y(x)"
	// та вказуємо тип його значень:
	ТаблицяГрафік->Columns->Add("Значення функції y(x)",
		double::typeid);
	do
	{
		y = MyFunc(x);
		//Формуємо рядок таблиці з округленням значень
		//функцією Math::Round() до 3 знаків після коми:
		Ряд["Значення x"] = Math::Round(x, 3,
			::MidpointRounding::AwayFromZero);
		Ряд["Значення функції y(x)"] = Math::Round(y, 3,
			::MidpointRounding::AwayFromZero);
		//Додаємо сформований рядок до таблиці:
		ТаблицяГрафік->Rows->Add(Ряд);
		//Створюємо наступний пустий рядок в таблиці:
		Ряд = ТаблицяГрафік->NewRow();
		++x;
	} while (x <= B);
	НабірДаних->Tables->Add(ТаблицяГрафік);
	return ТаблицяГрафік;
}
```

Функція-член `MyClass::ТаблицяЗначень()` повертає значення типу `DataTable^`, тобто таблицю значень залежності функції від її аргументу. Функція-член оголошена в інтерфейсі класу, тобто буде доступною користувачам класу.

```
public: //Інтерфейс класу
	//Об'явлення функцій-членів класу без параметрів
	DataTable^ ТаблицяЗначень();
```

В межах функції-члена оголошена локальна змінна Ряд класу `DataRow^` та ініціалізована, як новий рядок змінної-члену класу `ТаблицяГрафік`

```
DataRow^ Ряд = ТаблицяГрафік->NewRow();
```

В свою чергу змінна-член ТаблицяГрафік оголошена в закритій частині класу (private:), тобто є недоступною для користувачів класу. Доступ до цієї змінної мають лише члени класу та породжені від даного класу інші класи.

```
private: //Реалізація класу
     // Об'явлення та створення об'єкта "ТаблицяГрафік" класу DataTable:
     DataTable^ ТаблицяГрафік = gcnew DataTable();
```

На початку фінкції-члена MyClass::ТаблицяЗначень() до пустої таблиці даних ТаблицяГрафік додаються два нових стовпчики (поля даних) з іменами "Значення x" та "Значення функції y(x)":

```
//Додаємо в таблицю стовпчик з іменем "Значення x"
//та вказуємо тип його значень:
ТаблицяГрафік->Columns->Add("Значення x", double::typeid);
//Додаємо в таблицю стовпчик з іменем "Значення функції y(x)"
//та вказуємо тип його значень:
ТаблицяГрафік->Columns->Add("Значення функції y(x)", double::typeid);
```

Для нових стовпчиків таблиці також вказується тип даних, які вони зможуть зберігати:

```
Add("Значення x", double::typeid)
```

Надалі у циклі do while у змінну Ряд записуються значення x та розраховане за допомогою функції-члена MyFunc() відповідне значення y(x):

```
//Формуємо рядок таблиці з округленням значень
//функцією Math::Round() до 3 знаків після коми:
Ряд["Значення x"] = Math::Round(x, 3, ::MidpointRounding::AwayFromZero);
Ряд["Значення функції y(x)"] = Math::Round(y, 3,
     ::MidpointRounding::AwayFromZero);
```

після чого сформований рядок даних додається до таблиці ТаблицяГрафік та змінній Ряд присвоюється новий пустий рядок таблиці:

```
//Додаємо сформований рядок до таблиці:
ТаблицяГрафік->Rows->Add(Ряд);
//Створюємо наступний пустий рядок в таблиці:
```

```
Ряд = ТаблицяГрафік->NewRow();
```

По завершенні наповнення таблиці ТаблицяГрафік значеннями x та y дана таблиця додається до набору даних – змінної-члена НабірДаних:

```
НабірДаних->Tables->Add(ТаблицяГрафік);
```

По завершенні виконання функції MyClass::ТаблицяЗначень() вона повертає у місце свого виклику сформовану таблицю залежності y від x

```
return ТаблицяГрафік;
```

3.3.8 Створення функцій-членів для формування звіту та таблиці розрахунків

Оскільки змінні-члени Звіт, в якій зберігається сформований текстовий звіт по роботі програми, та ТаблицяІтерації, яка містить збережені результати розрахунку програми на кожній ітерації у вигляді таблиці даних, є закритими, тобто киристувач класу не має до них доступу, у клас включені дві відкриті функції-члена ТекстовийЗвіт() та ТаблицяРозрахунків():

```
public: //Інтерфейс класу
      //Об'явлення функцій-членів класу без параметрів:
      DataTable^ ТаблицяРозрахунків();
      String^ ТекстовийЗвіт();
```

Визначення цих функцій-членів класу наведене у лістингу 3.5.

Лістинг 3.5

```
String^ MyClass::ТекстовийЗвіт()
{
      return Звіт;
}

DataTable^ MyClass::ТаблицяРозрахунків()
{
      return ТаблицяІтерації;
}
```

Як видно з лістингу, дані функції-члени виконують лише задачу передавання вмісту закритих змінних-членів Звіт та ТаблицяІтерації за межі

класу. Тобто для отримання текстового звіту користувач повинен буде викликати функцію-член `ТекстовийЗвіт()` для створеного в тілі програми примірника класу `MyClass`, а для отримання результатів у табличному вигляді потрібно буде викликати функцію-член `ТаблицяРозрахунків()`.

Створення класу завершене. Для можливості його використання в програмному модулі потрібно підключити файл заголовку класу до файлу програмного коду за дапомогою директиви #include:

```
#include "MyMethod.h"
```

3.4 Програмування реакції на події

Після створення класу та підключення його файлу заголовку можна перейти до безпосереднього програмування реакції на події елементів інтерфейсу програмного модуля.

3.4.1 Обробник події розрахунків

Програмувати реакцію на події почнемо з пунктів головного меню. Процес розрахунку запрограмуємо як реакцію на подію `Click` пункту `розрахуватиToolStripMenuItem` головного меню (рис. 3.13).

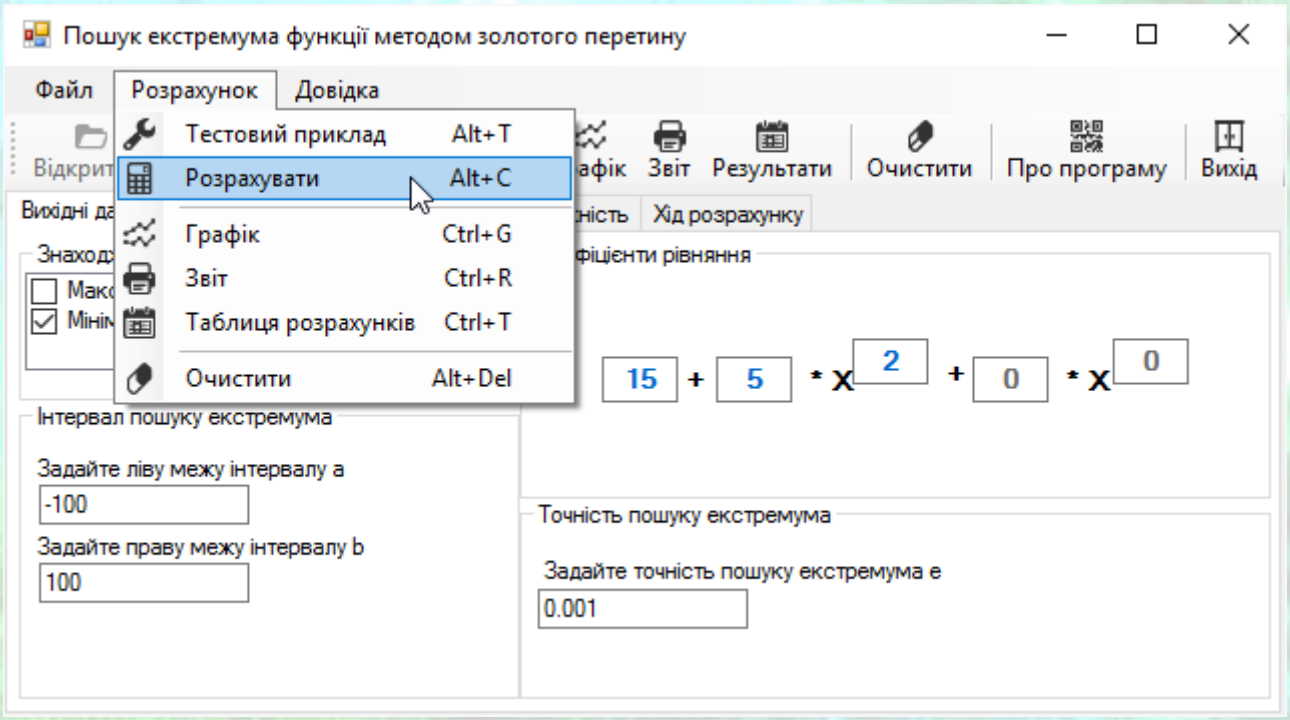


Рисунок 3.13 – Команда Розрахувати головного меню

Обробник даної події наведений в лістингу 3.6.

Лістинг 3.6

```
private: System::Void розрахуватиToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
     float k1, k2, k3, k4, k5, a, b, eps;
     bool mmaxx;
     try
     {
          if (!String::IsNullOrEmpty(textBox_a->Text)) a =
```

```
            float::Parse(textBox_a->Text);
      if (!String::IsNullOrEmpty(textBox_b->Text)) b =
            float::Parse(textBox_b->Text);
      if (!String::IsNullOrEmpty(textBox_e->Text)) eps =
            float::Parse(textBox_e->Text);
      if (checkedListBoxЕкстремум->GetItemChecked(1))
      {
            mmaxx = true;
      }
      else
      {
            mmaxx = false;
            checkedListBoxЕкстремум->SetItemChecked(0, true);
      }
      if ((textBox_k1->Text != "k1") && (textBox_k5->Text != "k5"))
      {
            if (!String::IsNullOrEmpty(textBox_k1->Text)) k1 =
                  float::Parse(textBox_k1->Text);
            if (!String::IsNullOrEmpty(textBox_k2->Text)) k2 =
                  float::Parse(textBox_k2->Text);
            if (!String::IsNullOrEmpty(textBox_k3->Text)) k3 =
                  float::Parse(textBox_k3->Text);
            if (!String::IsNullOrEmpty(textBox_k4->Text)) k4 =
                  float::Parse(textBox_k4->Text);
            if (!String::IsNullOrEmpty(textBox_k5->Text)) k5 =
                  float::Parse(textBox_k5->Text);
            Екстремум = gcnew MyClass(k1, k2, k3, k4, k5, a, b,
                  eps, mmaxx);
      }
      else if ((textBox_k1->Text != "k1") &&
            (textBox_k5->Text == "k5"))
      {
            if (!String::IsNullOrEmpty(textBox_k1->Text)) k1 =
                  float::Parse(textBox_k1->Text);
            if (!String::IsNullOrEmpty(textBox_k2->Text)) k2 =
                  float::Parse(textBox_k2->Text);
            if (!String::IsNullOrEmpty(textBox_k3->Text)) k3 =
                  float::Parse(textBox_k3->Text);
            Екстремум = gcnew MyClass(k1, k2, k3, a, b, mmaxx);
      }
      // Виклик функції розрахунку Method() та виведення
      //повідомлення з результатом, а також запитом на виведення
      //прміжних результатів:
```

```
            if ((MessageBox::Show(Екстремум->Method() +
                  "\n Вивести таблицю проміжних розрахунків?",
                  "Екстремум функції", MessageBoxButtons::YesNo,
                  MessageBoxIcon::Question)) == ::DialogResult::Yes)
                  таблицяРозрахункуToolStripMenuItem_Click(this, e);
      }
      catch (Exception^ Ситуація)
      {
            MessageBox::Show("Перевірте введені дані!\n" +
                  Ситуація->Message, "Помилкове введення",
                  MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
      }
}
```

Основне завдання цієї події – зчитування вхідних даних, які ввів користувач на сторінці Вихідні дані у вікні програми. При зчитуванні числових значень з властивості Text об'єктів класу TextBox виникає проблема перетворення типів даних, оскільки у властивості Text містяться лише дані рядкового типу.

Перетворення рядковий даних здійснюємо за допомогою функції Parse(). При цьому спочатку перевіряємо чи поле введення не пусте для уникнення подальших помилок при запуску розрахунку:

```
if ((!String::IsNullOrEmpty(textBox_a->Text)) a =
      float::Parse(textBox_a->Text);
```

В даній функції надалі перевіряємо яку опцію обрано в об'єкті checkedListBoxЕкстремум за допомогою функції GetItemChecked(). Якщо друга опція не відмічена, тоді встановлюємо відмітку на першій опції за допомогою функції SetItemChecked().

```
if (checkedListBoxЕкстремум->GetItemChecked(1))
{
      mmaxx = true;
}
      else
{
      mmaxx = false;
      checkedListBoxЕкстремум->SetItemChecked(0, true);
}
```

Змінна `mmaxx = true`, якщо шукаємо мінімум функції і `false`, якщо шукаємо максимум.

Після цього в умові

```
if ((textBox_k1->Text != "k1") && (textBox_k5->Text != "k5"))
```

ми перевіряємо чи задав користувач значення для коефіцієнтів k1 і k5. Якщо так, то припускаємо, що користувач увів значення для всіх коефіцієнтів рівняння і зчитуємо їх у відповідні змінні:

```
if (!String::IsNullOrEmpty(textBox_k1->Text)) k1 =
      float::Parse(textBox_k1->Text);
if (!String::IsNullOrEmpty(textBox_k2->Text)) k2 =
      float::Parse(textBox_k2->Text);
if (!String::IsNullOrEmpty(textBox_k3->Text)) k3 =
      float::Parse(textBox_k3->Text);
if (!String::IsNullOrEmpty(textBox_k4->Text)) k4 =
      float::Parse(textBox_k4->Text);
if (!String::IsNullOrEmpty(textBox_k5->Text)) k5 =
      float::Parse(textBox_k5->Text);
```

Надалі створюємо примірник нашого класу використовуючи конструктор з ініціалізацією всіх змінних-членів класу введеними користувачем даними.

```
Екстремум = gcnew MyClass(k1, k2, k3, k4, k5, a, b, eps, mmaxx);
```

Якщо умова `if ((textBox_k1->Text != "k1") && (textBox_k5 -> Text != "k5"))` не виконується, тоді перевіряємо наступну умову:

```
else if ((textBox_k1->Text != "k1") && (textBox_k5->Text == "k5"))
```

У разі її виконання зчитуємо лише значення для коефіцієнтів k1, k2, k3 та створюємо примірник класу за допомогою скороченого конструктора:

```
if (!String::IsNullOrEmpty(textBox_k1->Text)) k1 =
      float::Parse(textBox_k1->Text);
if (!String::IsNullOrEmpty(textBox_k2->Text)) k2 =
      float::Parse(textBox_k2->Text);
if (!String::IsNullOrEmpty(textBox_k3->Text)) k3 =
      float::Parse(textBox_k3->Text);
Екстремум = gcnew MyClass(k1, k2, k3, a, b, mmaxx);
```

Наступний рядок запускає безпосередньо знаходження екстремума функції:

```
if ((MessageBox::Show(Екстремум->Method() +
      "\n Вивести таблицю проміжних розрахунків?",
      "Екстремум функції", MessageBoxButtons::YesNo,
      MessageBoxIcon::Question)) == ::DialogResult::Yes)
      таблицяРозрахункуToolStripMenuItem_Click(this, e);
```

Тут формується рядок повідомлення зі знайденим екстремумом функції та запитом на виведення таблиці проміжних розрахунків

```
Екстремум->Method() + "\n Вивести таблицю проміжних розрахунків?",
"Екстремум функції"
```

який виводиться в діалоговому вікні MessageBox за допомогою метода Show з заголовком "Екстремум функції", типом діалогового вікна MessageBoxIcon::Question та двома кнопками Так та Ні - MessageBoxButtons::YesNo (рис. 3.14).

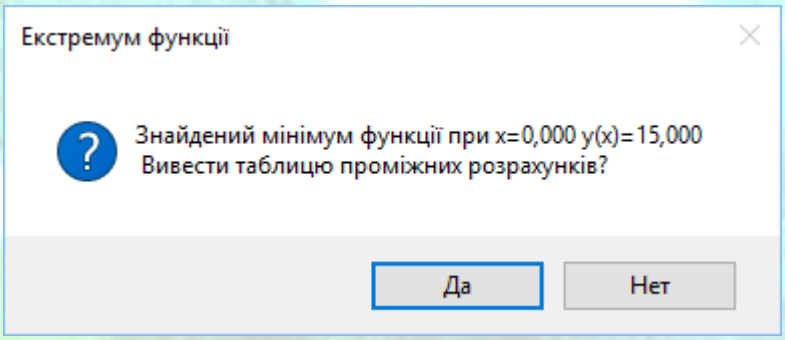


Рисунок 3.14 – Діалогове вікно з результатами розрахунку

В умові перевіряється чи натиснув користувач кнопку Так - ==::DialogResult::Yes. В цьому випадку викликається обробник події таблицяРозрахункуToolStripMenuItem_Click() який виводить таблицю проміжних розрахунків.

Зчитування даних з полів введення з перетворення типів даних - задача, яка потенційно може спричинити виключну ситуацію, якщо дані введені помилково, або відсутні чи не відповідного типу і не можуть бути перетворенні при зчитуванні у змінні. Тому оператори зчитування введених даних розміщують у блоці обробки виключних ситуацій try catch.

```
try
{

}
catch (Exception^ Ситуація)
{
      MessageBox::Show("Перевірте введені дані!\n" +
            Ситуація->Message, "Помилкове введення",
            MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
}
```

У разі виникнення помилки при введенні даних виведеться діалогове вікно з повідомленням (рис. 3.15)

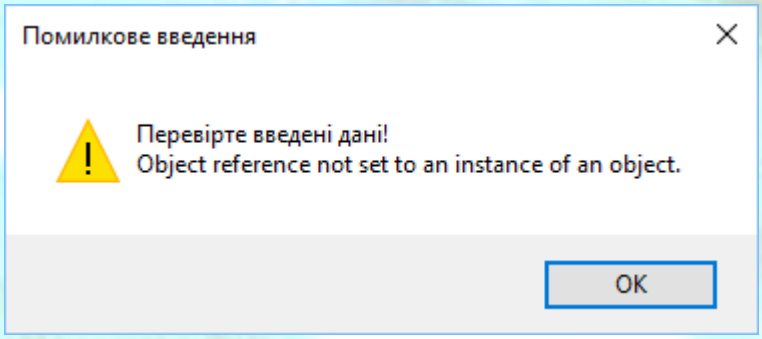


Рисунок 3.15 – Діалогове вікно з повідомленням про помилку введення

3.4.2 Обробник події тестового прикладу

В програмному модулі передбачений варіант запуску розрахунків із використанням тестового прикладу (рис. 3.16). Для цього призначений обробник події `тестовийПрикладToolStripMenuItem_Click()`, наведений в лістингу 3.7.

Лістинг 3.7

```
private: System::Void
тестовийПрикладToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
      Екстремум = gcnew MyClass();
      textBox_k1->Text = "15";
      textBox_k2->Text = "5";
      textBox_k3->Text = "2";
      textBox_k4->Text = "0";
      textBox_k5->Text = "0";
```

```
        textBox_a->Text = "-100";
        textBox_b->Text = "100";
        textBox_e->Text = "0.001";
        checkedListBoxЕкстремум->SetItemChecked(1, true);
        // Виклик функції розрахунку Method() та виведення повідомлення з
        // результатом, а також запитом на виведення прміжних результатів:
        if ((MessageBox::Show(Екстремум->Method() + "\n Вивести таблицю
              проміжних розрахунків?", "Екстремум функції",
              MessageBoxButtons::YesNo, MessageBoxIcon::Question)) ==
              ::DialogResult::Yes)
              таблицяРозрахункуToolStripMenuItem_Click(this, e);
}
```

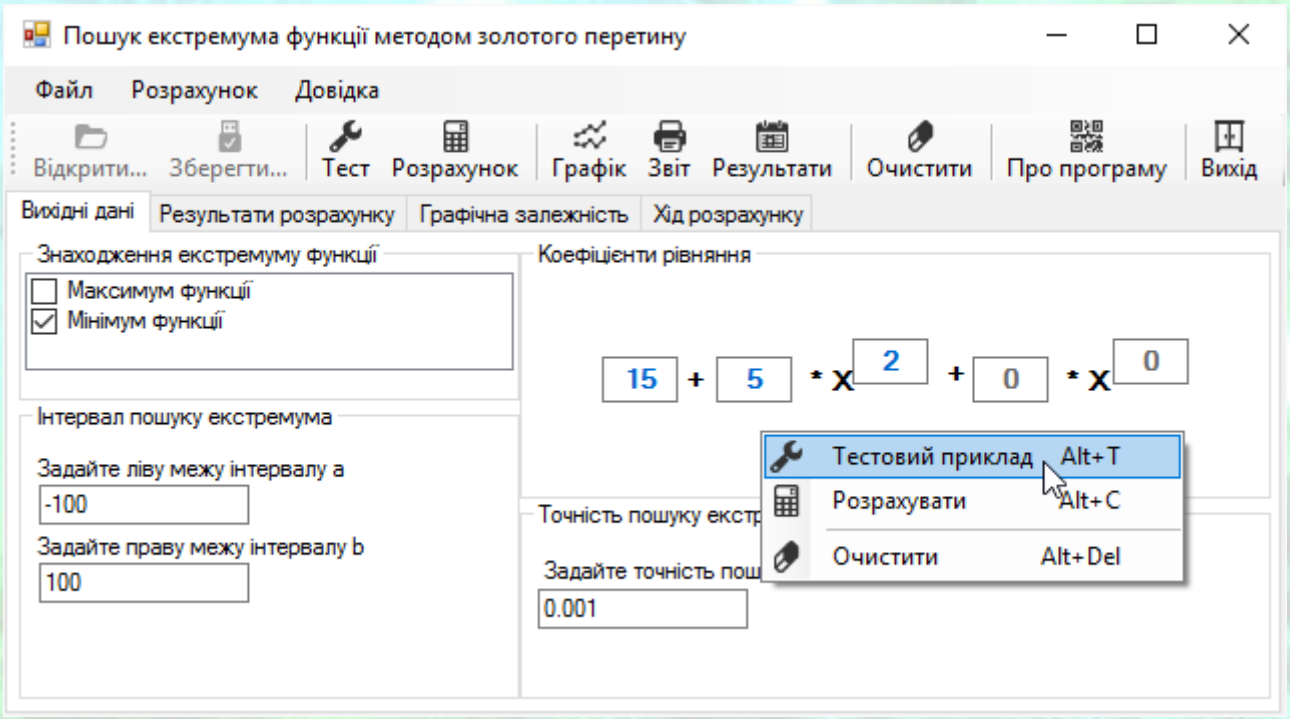


Рисунок 3.16 – Вікно з тестовим прикладом та контекстним меню

В даній процедурі створюється примірник нашого класу з використанням конструктора за замовчуванням `Екстремум = gcnew MyClass()`, а також у поля введення записуються вхідні дані для тестового прикладу. Після цього запускається пошук екстремума, як і в попередній процедурі.

3.4.3 Обробник події очищення даних

Для очищення полів введення вхідних даних, а також сторінки з текстовим звітом призначений обробник події очиститиToolStripMenuItem_Click() (лістинг 3.8).

Лістинг 3.8

```
private: System::Void очиститиToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
     //Перевірка чи активна перша вкладка:
     if (tabControlСторінки->SelectedIndex == 0)
     {
          // Очищення полів введення даних:
          textBox_k1->Text = "k1";
          textBox_k2->Text = "k2";
          textBox_k3->Text = "k3";
          textBox_k4->Text = "k4";
          textBox_k5->Text = "k5";
          textBox_a->Text = nullptr;
          textBox_b->Text = nullptr;
          textBox_e->Text = nullptr;
          checkedListBoxЕкстремум->SelectedIndex = -1;
          checkedListBoxЕкстремум->SetItemChecked(0, false);
          checkedListBoxЕкстремум->SetItemChecked(1, false);
     }
     else if (tabControlСторінки->SelectedIndex == 1)
     {
          видалитиToolStripMenuItem_Click(this, e);
     }
}
```

В залежності від того перша чи друга сторінка об’єкту tabControlСторінки активна, відбувається очищення полів вхідних даних чи текстового звіту.

3.4.4 Обробник події відкриття файлу

Для відкриття текстових та графічних файлів призначений обробник події відкритиФайлToolStripMenuItem_Click() (лістинг 3.9).

Лістинг 3.9

```
private: System::Void відкритиФайлToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
     //Перевірка чи активна третя вкладка:
     if (tabControlСторінки->SelectedIndex == 2)
     {//Тоді відкриваємо графічний файл
```

```
            //Створюємо об'єкт класу OpenFileDialog:
            OpenFileDialog ^ openFileDialog1 = gcnew OpenFileDialog();
            //Встановлюємо фільтр для відкриття файлів:
            openFileDialog1->Filter = L"Файли зображень |
                  *.bmp;*.jpg;*.gif; *.png";
            //Якщо користувач обрав файл і його ім'я не пусте:
            if (openFileDialog1->ShowDialog() == System::Windows::Forms
            ::DialogResult::OK && openFileDialog1->FileName->Length > 0)
            {
                  //Відкриваємо графічний файл в об'єкті pictureBox1:
                  pictureBox1->Image =
                        Image::FromFile(openFileDialog1->FileName);
                  //Розміщуємо ім'я відкритого файлу в заголовку вкладки:
                  tabPageГрафік->Text =
                        Path::GetFileName(openFileDialog1->FileName);
                  chartГрафік->Visible = false;
            }
      }
      //Активна друга вкладка:
      else if (tabControlСторінки->SelectedIndex == 1)
      {//Тоді відкриваємо файл *.rtf
            //Створюємо об'єкт класу OpenFileDialog:
            OpenFileDialog ^ openFileDialog1 = gcnew OpenFileDialog();
            //Встановлюємо фільтр для відкриття файлів:
            openFileDialog1->Filter = L"Текстові файли | *.rtf";
            //Якщо користувач обрав файл і його ім'я не пусте:
            if (openFileDialog1->ShowDialog() == System::Windows::Forms
            ::DialogResult::OK && openFileDialog1->FileName->Length > 0)
            {
                  //Відкриваємо файл формату RTF в об'єкті richTextBox1:
                  richTextBoxЗвіт->LoadFile(openFileDialog1->FileName);
                  //Розміщуємо ім'я відкритого файлу в заголовку вкладки:
                  tabPageРезультат->Text =
                        Path::GetFileName(openFileDialog1->FileName);
            }
      }
}
```

В процедурі перевіряється друга чи третя вкладка активна. Якщо активна сторінка Результати розрахунку (tabControlСторінки->SelectedIndex == 2), створюємо об'єкт класу OpenFileDialog:

```
OpenFileDialog ^ openFileDialog1 = gcnew OpenFileDialog();
```

Після цього встановлюємо фільтр для відкриття графічних файлів у діалоговому вікні:

```
openFileDialog1->Filter = L"Файли зображень | *.bmp;*.jpg;*.gif; *.png";
```

Надалі перевіряємо чи натиснув користувач кнопку Відкрити в діалоговому вікні відкриття файлу та чи не пусте поле імені файлу:

```
if (openFileDialog1->ShowDialog() == System::Windows::Forms
    ::DialogResult::OK && openFileDialog1->FileName->Length > 0)
```

У разі виконання цієї умови відкриваємо обраний користувачем графічний файл у об'єкті pictureBox1:

```
pictureBox1->Image = Image::FromFile(openFileDialog1->FileName);
```

Встановлюємо у заголовку вкладки ім'я відкритого файлу:

```
tabPageГрафік->Text = Path::GetFileName(openFileDialog1->FileName);
```

Для цього використовуємо функцію Path::GetFileName(), яка повертає ім'я файлу з рядка із повним ім'ям файлу, що зберігається у властивості FileName об'єкту openFileDialog1.

Для того, щоб графічний файл став видимим, приховуємо об'єкт chartГрафік, який розташований над об'єктом pictureBox1 на сторінці tabPageГрафік:

```
chartГрафік->Visible = false;
```

У разі, якщо активна сторінка Результати розрахунку, встановлюємо у діалоговому вікні фільтр для відкриття файлів формату *.rtf:

```
openFileDialog1->Filter = L"Текстові файли | *.rtf";
```

Тепер відкриваємо файл у об'єкті richTextBoxЗвіт за допомогою метода LoadFile та встановлюємо ім'я відкритого файлу як заголовок сторінки:

```
richTextBoxЗвіт->LoadFile(openFileDialog1->FileName);
tabPageРезультат->Text = Path::GetFileName(openFileDialog1->FileName);
```

3.4.5 Обробник події збереження файлу

Для збереження файлів у програмному модулі передбачений обробник події `зберегтиФайлToolStripMenuItem_Click()` (лістинг 3.10).

Лістинг 3.10

```
private: System::Void зберегтиФайлToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
     //Перевірка чи активна перша вкладка:
     if (tabControlСторінки->SelectedIndex == 2)
     {//Тоді зберігаємо графічний файл
          // Відобразити вікно SaveFileDialog для збереження зображення
          SaveFileDialog ^ saveFileDialog1 = gcnew SaveFileDialog();
          saveFileDialog1->Filter = "Зображення JPeg|*.jpg|
               Зображення Bitmap|*.bmp|Зображення Gif|*.gif";
          saveFileDialog1->Title = "Зберегти файл зображення";
          saveFileDialog1->ShowDialog();
          // Якщо ім'я файлу не пустий рядок, зберегти зображення:
          if (saveFileDialog1->FileName != "")
          {
               // Зберегти зображення через System::IO::FileStream,
               // створений методом OpenFile
               ::IO::FileStream ^ fs = safe_cast<System::IO
                    ::FileStream^>(saveFileDialog1->OpenFile());
               // Запам'ятовування графічного формату файлу
               //System::Drawing::Imaging::ImageFormat:
               ::Imaging::ImageFormat^ ГрафічнийФормат;
               // Зберігаємо зображення у відповідному форматі
               //ImageFormat на основі типу файлу,
               //який міститься у властивості FilterIndex
               switch (saveFileDialog1->FilterIndex)
               {
               case 1:
                    ГрафічнийФормат = ::Imaging::ImageFormat::Jpeg;
                    break;
               case 2:
                    ГрафічнийФормат = ::Imaging::ImageFormat::Bmp;
                    break;
               case 3:
                    ГрафічнийФормат = ::Imaging::ImageFormat::Gif;
                    break;
```

```
                }
                // Перевіряємо чи створена діаграма (графік):
                if (chartГрафік->Visible == true)
                {
                      // Зберігаємо діаграму:
                      chartГрафік->SaveImage(fs, ГрафічнийФормат);
                }
                // Інакше зберігаємо відкритий малюнок:
                else pictureBox1->Image->Save(fs, ГрафічнийФормат);
                //Розміщуємо ім'я файлу в заголовку вкладки:
                tabControlСторінки->SelectedTab->Text =
                      Path::GetFileName(saveFileDialog1->FileName);
                fs->Close();
          }
    }
    //Активна друга вкладка:
    else if (tabControlСторінки->SelectedIndex == 1)
    {//Тоді зберігаємо файл *.rtf
          //Створюємо об'єкт класу SaveFileDialog:
          SaveFileDialog ^ saveFileDialog1 = gcnew SaveFileDialog();
          //Встановлюємо фільтр для збереження файлів:
          saveFileDialog1->Filter = L"Текстові файли | *.rtf";
          //Якщо користувач обрав файл і його ім'я не пусте:
          if (saveFileDialog1->ShowDialog() ==
                System::Windows::Forms::DialogResult::OK &&
                saveFileDialog1->FileName->Length > 0)
          {
                //Зберігаємо файл формату RTF з об'єкту richTextBox1:
                richTextBoxЗвіт->SaveFile(saveFileDialog1->FileName);
                //Розміщуємо ім'я файлу в заголовку вкладки:
                tabPageРезультат->Text =
                      Path::GetFileName(saveFileDialog1->FileName);
          }
    }
}
```

Аналогічно як і в попередньому обробнику події відкритиФайлToolStripMenuItem_Click(), в обробнику події збереження файлу перевіряється яка зі сторінок об’єкту tabControlСторінки активна. У разі якщо активною є сторінка Графічна залежність, спочатку створюється об’єкт класу SaveFileDialog, потім задається фільтр графічних файлів для

діалогового вікна збереження файлу, а також заголовок вікна. Після чого діалогове вікно виводиться на екран (рис. 3.17):

```
SaveFileDialog ^ saveFileDialog1 = gcnew SaveFileDialog();
saveFileDialog1->Filter = "Зображення JPeg|*.jpg|
     Зображення Bitmap|*.bmp|Зображення Gif|*.gif";
saveFileDialog1->Title = "Зберегти файл зображення";
saveFileDialog1->ShowDialog();
```

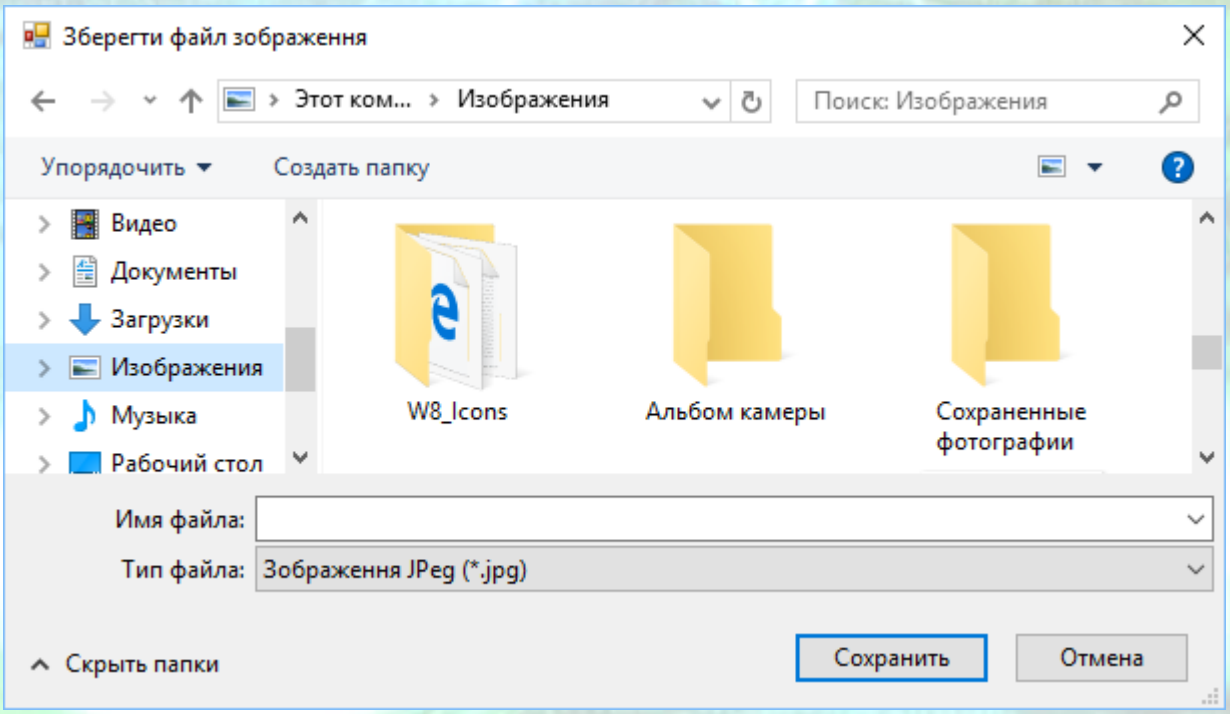


Рисунок 3.17 – Створене діалогове вікно збереження графічного зображення

Якщо ім'я файлу не пустий рядок, тоді необхідно зберегти зображення.

```
::IO::FileStream ^ fs = safe_cast<System::IO
     ::FileStream^>(saveFileDialog1->OpenFile());
::Imaging::ImageFormat^ ГрафічнийФормат;
```

Для збереження зображення у відповідному форматі за допомогою змінної ГрафічнийФормат типу ImageFormat запам’ятовуємо формат файлу, який був обраний користувачем у вікні збереження файлу.

```
switch (saveFileDialog1->FilterIndex)
{
     case 1:
          ГрафічнийФормат = ::Imaging::ImageFormat::Jpeg;
          break;
```

```
        case 2:
            ГрафічнийФормат = ::Imaging::ImageFormat::Bmp;
            break;
        case 3:
            ГрафічнийФормат = ::Imaging::ImageFormat::Gif;
            break;
}
```

Якщо видимий об'єкт для побудови графіка chartГрафік, тоді необхідно зберегти побудований графік у вигляді файлу зображення. У іншому разі зберігаємо зображення відкрите у об'єкті pictureBox1.

```
if (chartГрафік->Visible == true)
{
    // Зберігаємо діаграму:
    chartГрафік->SaveImage(fs, ГрафічнийФормат);
}
// Інакше зберігаємо відкритий малюнок:
else pictureBox1->Image->Save(fs, ГрафічнийФормат);
```

У випадку коли активна сторінка Результати розрахунку необхідно створити об'єкт класу SaveFileDialog, задати фільтр для файлів *.rtf та відобразити діалогове вікно збереження файлу. Якщо поле імені файлу не пусте, зберігаємо вміст об'єкту richTextBoxЗвіт у файлі *.rtf.

```
richTextBoxЗвіт->SaveFile(saveFileDialog1->FileName);
```

Також встановлюємо ім'я збереженого файлу на заголовок вкладки.

```
tabControlСторінки->SelectedTab->Text =
    Path::GetFileName(saveFileDialog1->FileName);
```

3.4.6 Обробник події побудови графіку

Щоб побудувати графічну залежність досліджуваної функції в програмному модулі передбачено обробник події графікToolStripMenuItem_Click() (лістинг 3.11).

Лістинг 3.11

```
private: System::Void графікToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
      chartГрафік->DataSource = Екстремум->ТаблицяЗначень();
      // Прив'язка графіка до джерела даних:
      chartГрафік->DataBind();
      // На горизонтальній осі відкладаємо значення x:
      chartГрафік->Series["Series1"]->XValueMember = "Значення x";
      // А по вертикальній осі відкладаємо значення z:
      chartГрафік->Series["Series1"]->YValueMembers
            = "Значення функції y(x)";
      // Задаємо тип діаграми - лінійна:
      chartГрафік->Series["Series1"]->ChartType = SeriesChartType::Line;
      // Тип діаграми може бути іншим, наприклад: Pie, Column і ін.
      chartГрафік->Series["Series1"]->Color = Color::Aqua;
      // Легенду на графіку не відображаємо:
      chartГрафік->Series["Series1"]->IsVisibleInLegend = false;
      chartГрафік->Visible = true;
      tabControlСторінки->SelectedIndex = 2;
}
```

Властивості DataSource об’єкту chartГрафік присвоюється результат виконання функції-члена ТаблицяЗначень(), яка повертає значення типу DataTable.

```
chartГрафік->DataSource = Екстремум->ТаблицяЗначень();
```

Після цього прив’язуємо дані об’єкту chartГрафік класу Chart до даних з джерела даних за допомогою методу Chart::DataBind():

```
chartГрафік->DataBind();
```

Надалі для набору даних "Series1" об’єкту chartГрафік встановлюємо підписи для значень по осі ординат та осі абсцис:

```
chartГрафік->Series["Series1"]->XValueMember = "Значення x";
chartГрафік->Series["Series1"]->YValueMembers = "Значення функції y(x)";
```

Типи діаграм зберігаються в переліку SeriesChartType. З даного переліку обираємо тип діаграми – графік (Line):

```
chartГрафік->Series["Series1"]->ChartType = SeriesChartType::Line;
```

Колір діаграми задаємо за допомогою властивості Color:

```
chartГрафік->Series["Series1"]->Color = Color::Aqua;
```

Наостанок об'єкт chartГрафік робимо видимим та активною задаємо сторінку Графічна залежність – третю сторінку об'єкту tabControlСторінки (рис. 3.18).

```
chartГрафік->Visible = true;
tabControlСторінки->SelectedIndex = 2;
```

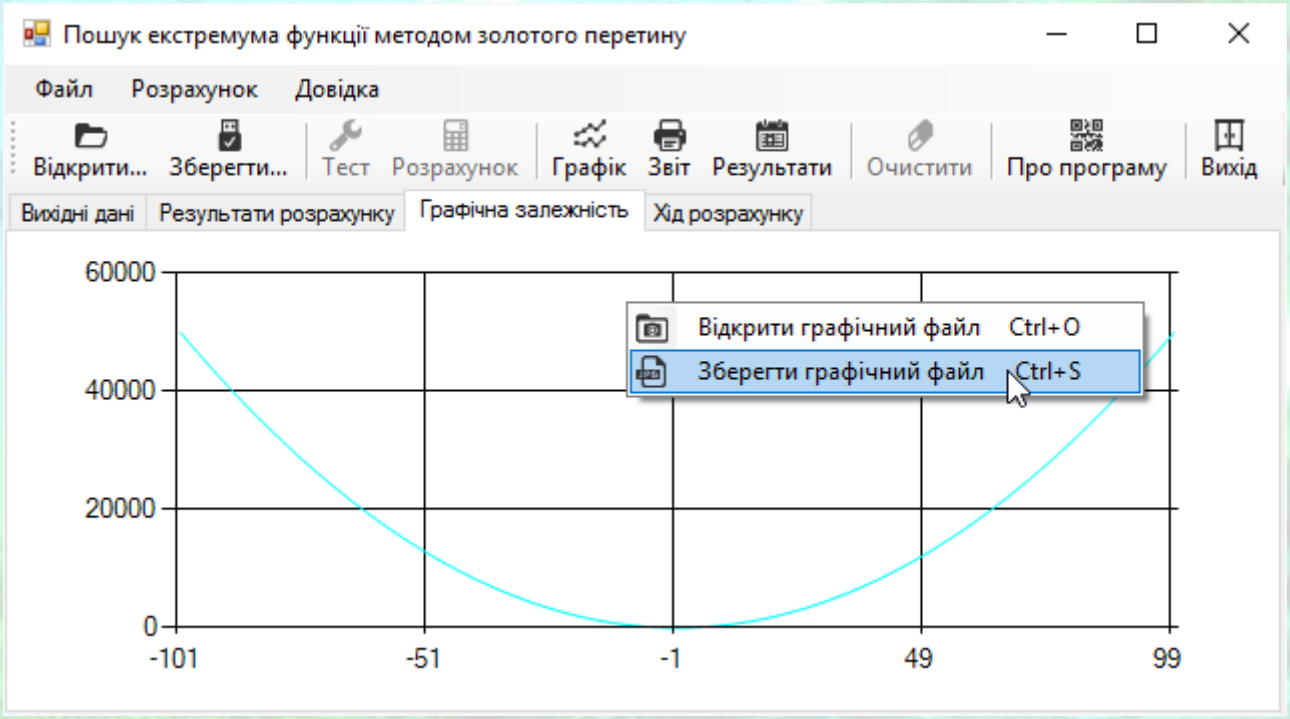


Рисунок 3.18 – Графічна залежність функції та контекстне меню

3.4.7 Обробник події створення звіту

При виклику команди створення звіту Звіт з групи команд Розрахунок головного меню відбувається виведення текстового звіту в об'єкті richTextBoxЗвіт (лістинг 3.12)

Лістинг 3.12

```
private: System::Void звітToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e) {
```

```
        // Виведення результатів розрахунку в текстове поле richTextBoxЗвіт
        richTextBoxЗвіт->AppendText(Екстремум->ТекстовийЗвіт());
        tabControlСторінки->SelectedIndex = 1;
        tabPageРезультат->Text = "Результати розрахунку";
}
```

Оскільки весь текстовий звіт формується у функції-члені `ТекстовийЗвіт()` нашого класу у вигляді об'єкту типу `String^`, то виведення звіту у полі `richTextBoxЗвіт` виконується за допомогою метода `AppendText()`, який додає текст в кінець текстового поля.

```
richTextBoxЗвіт->AppendText(Екстремум->ТекстовийЗвіт());
```

В якості аргументу для метода `AppendText()` передається функція-член `ТекстовийЗвіт()`.

Після цього поточна сторінка змінюється на сторінку `tabPageРезультат`, яка має індекс 1 в об'єкті `tabControlСторінки`:

```
tabControlСторінки->SelectedIndex = 1;
```

В якості заголовку сторінки встановлюємо значення `"Результати розрахунку"`:

```
tabPageРезультат->Text = "Результати розрахунку";
```

Це потрібно на той випадок, коли на сторінці `tabPageРезультат` користувач відкривав текстовий файл і заголовок сторінки був змінений на ім'я відкритого файлу (рис. 3.19).

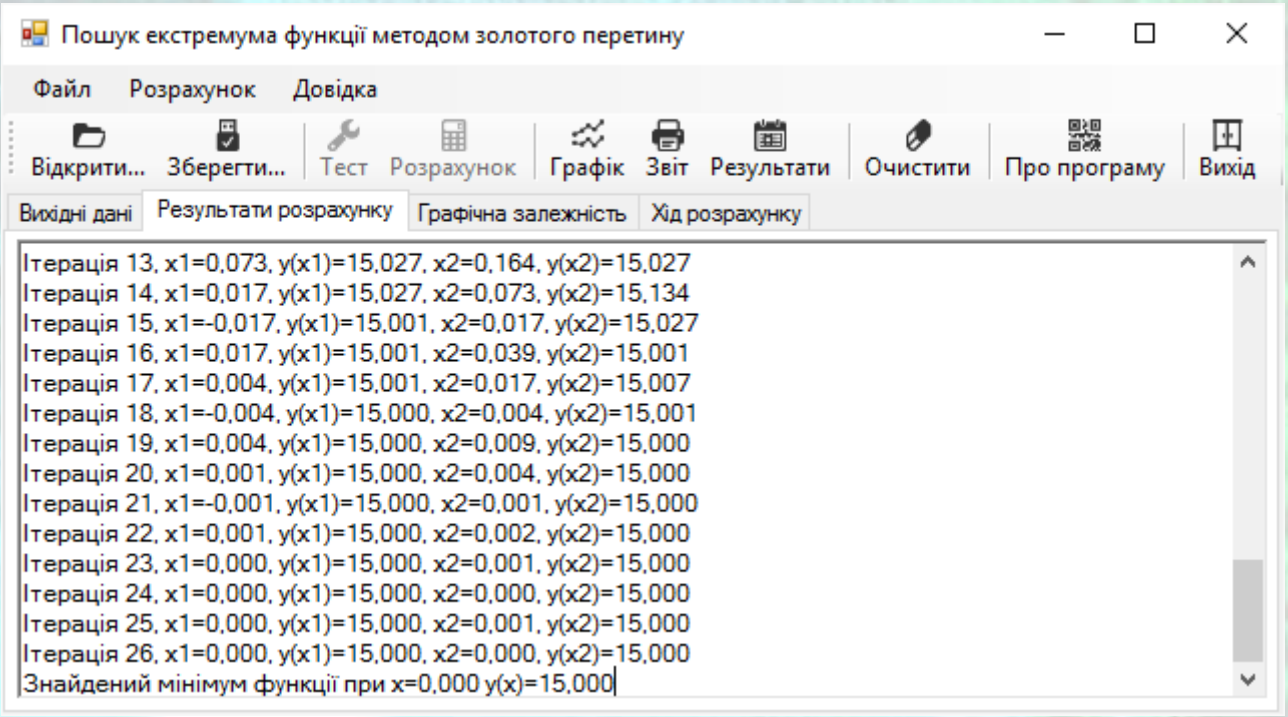


Рисунок 3.19 – Текстовий звіт за результатами розрахунку

3.4.8 Обробник події виведення результатів в табличному вигляді

При виклику команди Таблиця розрахунків з групи команд Розрахунок головного меню відбувається виведення результатів розрахунку в табличному вигляді в об'єкті dataGridViewІтерації класу DataGridView (лістинг 3.13)

Лістинг 3.13

```
private: System::Void таблицяРозрахункуToolStripMenuItem_Click(System::
Object^ sender, System::EventArgs^ e) {
    // Виведення результатів розрахунку в таблицю dataGridViewІтерації
    dataGridViewІтерації->DataSource = Екстремум->ТаблицяРозрахунків();
    tabControlСторінки->SelectedIndex = 3;
}
```

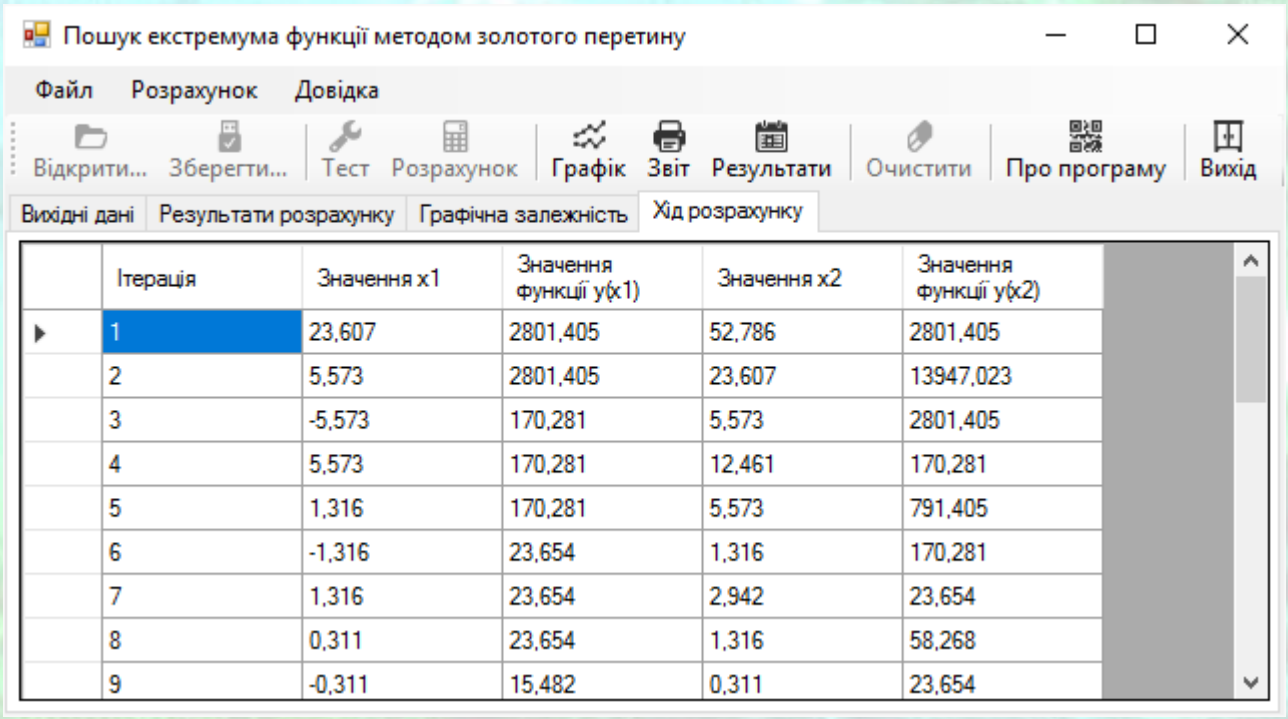
Таблиця результатів розрахунку створюється функцією-членом ТаблицяРозрахунків() класу MyClass, яка повертає значення типу DataTable^. Згідно цього об'єкту dataGridViewІтерації в якості джерела даних присвоюємо результат виконання функції ТаблицяРозрахунків():

```
dataGridViewІтерації->DataSource = Екстремум->ТаблицяРозрахунків();
```

Надалі робимо активною сторінку з отриманою таблицею результатів розрахунку (рис. 3.20):

```
tabControlСторінки->SelectedIndex = 3;
```

З наведеного на рис. 3.20 вікна програми видно, що в об'єкті класу TabControl відобразилась вкладка з назвою Хід розрахунку, яка є четвертою в об'єкті tabControlСторінки та має порядковий індекс 3. Присвоєння властивості SelectedIndex об'єкту TabControl порядкового номеру відповідної сторінки робить дану сторінку поточною та відображає її у вікні програми.



	Ітерація	Значення x1	Значення функції y(x1)	Значення x2	Значення функції y(x2)
▶	1	23,607	2801,405	52,786	2801,405
	2	5,573	2801,405	23,607	13947,023
	3	-5,573	170,281	5,573	2801,405
	4	5,573	170,281	12,461	170,281
	5	1,316	170,281	5,573	791,405
	6	-1,316	23,654	1,316	170,281
	7	1,316	23,654	2,942	23,654
	8	0,311	23,654	1,316	58,268
	9	-0,311	15,482	0,311	23,654

Рисунок 3.20 – Деактивація неактивних команд на панелі інструментів

3.4.9 Обробник події активації та деактивації команд головного меню та панелі інструментів

В залежності від того, яка сторінка об'єкту tabControlСторінки активна – Вихідні дані, Результати розрахунку, Графічна залежність чи Хід розрахунку, можливе виконання одних команд меню та неможливе виконання інших. Наприклад, якщо активна сторінка Хід розрахунку, в програмі не передбачене відкриття чи збереження даних на цій сторінці, відповідно ці

команди в головному меню та на панелі інструментів повинні бути неактивними (рис. 3.20).

Контроль за активацією та деактивацією команд в головному меню та на панелі інструментів здійснюється в обробнику події зміни активної сторінки в об'єкті tabControlСторінки (лістинг 3.14).

Лістинг 3.14

```
private: System::Void
tabControlСторінки_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
      //Перевірка чи активна перша вкладка:
      if (tabControlСторінки->SelectedIndex == 0)
      {
            toolStripButtonЗберегти->Enabled = false;
            зберегтиФайлToolStripMenuItem->Enabled = false;
            toolStripButtonТест->Enabled = true;
            тестовийПрикладToolStripMenuItem->Enabled = true;
            toolStripButtonРозрахувати->Enabled = true;
            розрахуватиToolStripMenuItem->Enabled = true;
            toolStripButtonОчистити->Enabled = true;
            очиститиToolStripMenuItem->Enabled = true;
            toolStripButtonВідкрити->Enabled = false;
            відкритиФайлToolStripMenuItem->Enabled = false;
      }
      //Перевірка чи активна друга вкладка:
      else if (tabControlСторінки->SelectedIndex == 1)
      {
            toolStripButtonЗберегти->Enabled = true;
            зберегтиФайлToolStripMenuItem->Enabled = true;
            toolStripButtonТест->Enabled = false;
            тестовийПрикладToolStripMenuItem->Enabled = false;
            toolStripButtonРозрахувати->Enabled = false;
            розрахуватиToolStripMenuItem->Enabled = false;
            toolStripButtonОчистити->Enabled = true;
            очиститиToolStripMenuItem->Enabled = true;
            toolStripButtonВідкрити->Enabled = true;
            відкритиФайлToolStripMenuItem->Enabled = true;
      }
      //Перевірка чи активна третя вкладка:
      else if (tabControlСторінки->SelectedIndex == 2)
      {
```

```
            toolStripButtonЗберегти->Enabled = true;
            зберегтиФайлToolStripMenuItem->Enabled = true;
            toolStripButtonТест->Enabled = false;
            тестовийПрикладToolStripMenuItem->Enabled = false;
            toolStripButtonРозрахувати->Enabled = false;
            розрахуватиToolStripMenuItem->Enabled = false;
            toolStripButtonОчистити->Enabled = false;
            очиститиToolStripMenuItem->Enabled = false;
            toolStripButtonВідкрити->Enabled = true;
            відкритиФайлToolStripMenuItem->Enabled = true;
      }
      //Якщо активна четверта вкладка:
      else
      {
            toolStripButtonЗберегти->Enabled = false;
            зберегтиФайлToolStripMenuItem->Enabled = false;
            toolStripButtonТест->Enabled = false;
            тестовийПрикладToolStripMenuItem->Enabled = false;
            toolStripButtonРозрахувати->Enabled = false;
            розрахуватиToolStripMenuItem->Enabled = false;
            toolStripButtonОчистити->Enabled = false;
            очиститиToolStripMenuItem->Enabled = false;
            toolStripButtonВідкрити->Enabled = false;
            відкритиФайлToolStripMenuItem->Enabled = false;
      }
}
```

Даний обробник події при зміні сторінки в об’єкті tabControlСторінки робить певні команди меню та панелі інструментів неактивними, а інші активними змінюючи значення їх властивості Enabled.

3.4.10 Обробники подій редагування тексту

Для сторінки tabPageРезультат передбачене власне контекстне меню contextMenuStripРезультати, яке пов’язане з даним об’єктом за допомогою властивості contextMenuStrip (рис. 3.21). В даному контекстному меню розміщені команди редагування тексту, обробники яких наведені в лістингу 3.15.

Лістинг 3.15

```
private: System::Void виділитиВсеToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
```

```
        // Виділяє весь текст
        richTextBoxЗвіт->SelectAll();
    }

    private: System::Void скопіюватиToolStripMenuItem_Click(System::Object^
    sender, System::EventArgs^  e) {
        if (richTextBoxЗвіт->SelectionLength > 0)
        {
            // Копіює виділений текст в буфер обміну
            richTextBoxЗвіт->Copy();
        }
    }

    private: System::Void вирізатиToolStripMenuItem_Click(System::Object^
    sender, System::EventArgs^  e) {
        // Перевіряємо чи є виділений текст
        if (!richTextBoxЗвіт->SelectedText->Equals(""))
        {
            // Вирізає виділений текст та вставляє його в буфер обміну
            richTextBoxЗвіт->Cut();
        }
    }

    private: System::Void видалитиToolStripMenuItem_Click(System::Object^
    sender, System::EventArgs^  e) {
        // Очищує текст (видаляє) в об'єкті richTextBox1
        richTextBoxЗвіт->Clear();
    }

    private: System::Void вставитиToolStripMenuItem_Click(System::Object^
    sender, System::EventArgs^  e) {
        // Вставляє текст з буферу обміну
        richTextBoxЗвіт->Paste();
    }

    private: System::Void відмінитиToolStripMenuItem_Click(System::Object^
    sender, System::EventArgs^  e) {
        if (richTextBoxЗвіт->CanUndo == true)
        {
            // Відміняє останню операцію
            richTextBoxЗвіт->Undo();
            // Очищує буфер undo від збереженої попередньої операції
            richTextBoxЗвіт->ClearUndo();
```

```
        }
}
```

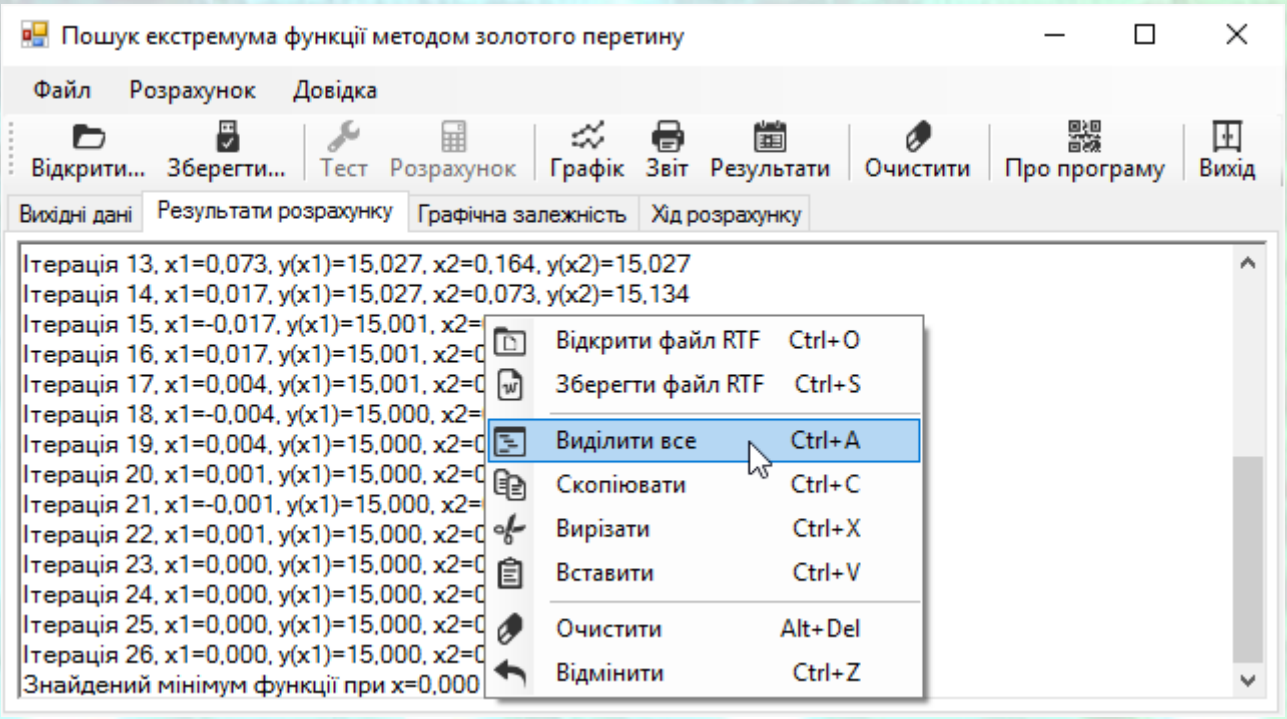


Рисунок 3.21 – Контекстне меню на сторінці Результати розрахунку

3.4.11 Обробники подій **Про програму** та **Вихід**

В головному меню програми в групі команд Довідка є команда Про програму, яка виводить діалогове вікно з короткою інформаціює про програму (рис. 3.22).

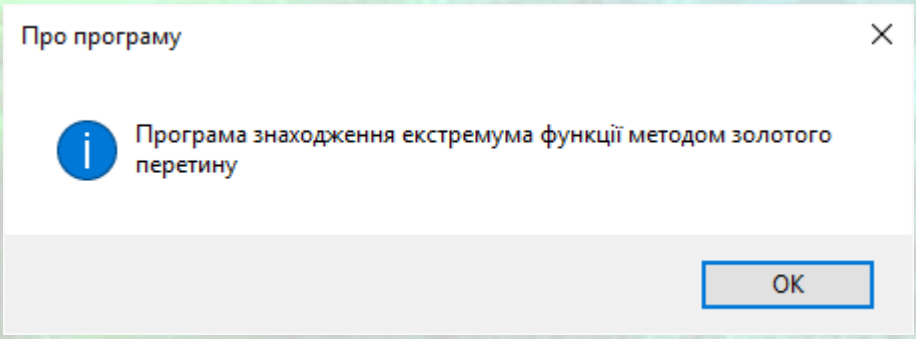


Рисунок 3.22 – Діалогове вікно Про програму

Дану команду можна викликати клавішею F1 завдяки встановленому значенню властивості ShortcutKeys для об'єкта проПрограмуToolStripMenuItem.

Діалогове вікно виводиться методом Show об'єкта MessageBox, якому передано чотири параметри: текст повідомлення, заголовок вікна, набір кнопок діалогового вікна та тип діалогового вікна (лістинг 3.16).

Закриття програми здійснюється викликом обробника події вихідToolStripMenuItem_Click() в якому викликається метод Close() для форми.

Лістинг 3.16

```
private: System::Void проПрограмуToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
     MessageBox::Show("Програма знаходження екстремума функції методом
           золотого перетину", "Про програму", MessageBoxButtons::OK,
          MessageBoxIcon::Information);
}

private: System::Void вихідToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
     Close();//Закриття форми
}
```

На цьому створення обробників подій виклику команд меню завершено. Оскільки команди панелі інструментів та деякі команди контекстних меню повторюють команди головного меню, то для них окремі обробники подій не створюються. Для того, щоб, наприклад, при виклику команд панелі інструментів виконувались необхідні операції, на сторінці Події (Events) вікна Властивості (Properties) для події Click з випадаючого списку обирається вже існуючий необхідний метод обробки даної події.

3.4.12 Обробники подій форми

Окрім запрограмованих подій виклику команд меню та панелей інструментів, в нашому програмному модулі існує кілька обробників подій форми, які будуть вікликатися при виникненні певної події (лістинг 3.17).

Лістинг 3.17

```
private: System::Void MyForm_Load(System::Object^  sender,
System::EventArgs^  e) {
     tabControlСторінки_SelectedIndexChanged(this, e);
```

```
}

private: System::Void MyForm_Resize(System::Object^  sender,
System::EventArgs^  e) {
     //Вирівнювання ширини панелей в межах форми:
     panelЕкстремумІнтервал->Width = panelКоефіцієнтиТочність->
          Width - 170;
}

private: System::Void MyForm_FormClosing(System::Object^  sender,
System::Windows::Forms::FormClosingEventArgs^  e) {
     //Якщо натиснуте No, не закривати форму:
     if ((MessageBox::Show("Вийти з програми?", "Вихід",
          MessageBoxButtons::YesNo, MessageBoxIcon::Question)) ==
          ::DialogResult::No) e->Cancel = true;
}
```

Так при завантаженні форми виникає стандартна подія форми MyForm_Load(). В даній події у нас розташований виклик іншої події - tabControlСторінки_SelectedIndexChanged() для того, щоб привести у відповідність доступність команд меню та панелі інструментів перед відкриттям вікна.

Наступна подія форми, яку ми використала, це подія MyForm_Resize(). Дана подія викликається кожного разу при зміні розміру форми (вікна програми). Цю подію зазвичай використовують для перемальовування елементів інтерфейсу і контролю їх розташування та розмірів. В нашому випадку ми змінюємо ширину об'єкту panelЕкстремумІнтервал в залежності від зміни ширини іншого об'єкту – panelКоефіцієнтиТочність:

```
//Вирівнювання ширини панелей в межах форми:
panelЕкстремумІнтервал->Width = panelКоефіцієнтиТочність->Width - 170;
```

Подія FormClosing() викликається безпосередньо перед закриттям форми. В цю подію розміщують оператори, які мають виконатись безпосередньо перед закриттям форми. Наприклад, це може бути запит на збереження файлу перед виходом з програми або запит на закриття програми.

В нашому випадку ми виводимо діалогове вікно на екран з запитом про вихід з програми (рис. 3.23).

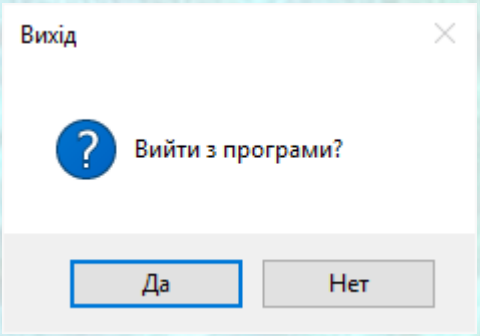


Рисунок 3.23 – Діалогове вікно виходу з програми

В діалоговому вікні-запиті (`MessageBoxIcon::Question`) присутні дві кнопки `Так` та `Ні` (`MessageBoxButtons::YesNo`). Якщо користувач натисне кнопку `Ні`, діалогове вікно поверне результат `::DialogResult::No`. В такому випадку властивості `Cancel` параметру `e` типу `System::Windows::Forms::FormClosingEventArgs^` ми присвоюємо значення `true`:

```
if ((MessageBox::Show("Вийти з програми?", "Вихід",
     MessageBoxButtons::YesNo, MessageBoxIcon::Question)) ==
     ::DialogResult::No) e->Cancel = true;
```

Закриття вікна буде відмінене.

4 КОНТРОЛЬНІ ПИТАННЯ

1) Що таке об'єктно-орієнтоване програмування? Що таке властивості, методи та конструктори в об'єктно-орієнтованому програмуванні?
2) Що таке інкапсуляція даних?
3) Що таке абстракція даних? Що таке абстрактний тип даних?
4) Що таке інтерфейс та реалізація класу?
5) Що таке інкапсуляція? Що таке специфікатори доступу?
6) У чому різниця між використанням ключових слів `struct` та `class`?
7) Що таке внутрішньокласовий ініціалізатор? Що таке змінні-члени класу?
8) Що таке функції-члени? Де та як вони об’являються та визначаються?
9) Що таке область видимості класу? Наведіть приклад визначення функції-члену поза тілом класу.
10) Для чого використовується параметр `this`? Як звернутися до члену об’єкту класу? Наведіть приклади.
11) Як підключити допоміжні функції для використання класом без їх включення у клас?
12) Що таке оператор доступу до члену та оператор області видимості?
13) Що таке конструктор, для чого він використовується, чим відрізняються конструктори одного класу?
14) Що таке стандартний конструктор? Що таке синтезований стандартний конструктор? Як об’явити стандартний конструктор?
15) Що таке перелік ініціалізації конструктора? Для чого він потрібен?
16) Де потрібно визначати конструктор? Із чого складається конструктор? Наведіть приклади об’явлення та визначення конструкторів. Що таке деструктор класу?
17) Що таке дружні відносини? Як об’явити дружню функцію? Як зробити дружню функцію доступною для користувачів класу?
18) Для чого потрібні файли заголовку? Як підключити заголовок до програми? Як організувати захист заголовку?
19) Що таке змінні препроцесора? Які правила створення імен змінних препроцесору та файлів заголовку загальноприйняті? Як працює директива `#include`?
20) Що таке клас в середовищі CLR? Що таке примірник та об’єкт класу?

21) Що таке спадковість (наслідування) класів?

22) Що таке компоненти? Чим вони відрізняються від звичайних класів?

23) Що таке властивості та обробники подій компонентів?

24) Опишіть призначення атрибутів доступу до членів класу середовища CLR - `public`, `private` та `protected`.

25) Що таке поліморфізм? Наведіть приклади. Для чого використовується атрибут `virtual`?

26) В чому відмінність статичної і динамічної пам'яті? Що таке "купа" (heap)? В чому відмінність керованої і некерованої купи?

27) Для чого використовуються функції `malloc()` і `free()`? Для чого потрібні оператори `new` і `delete`?

28) В чому різниця між рідними (`native`) класами та структурами та керованими (`managed`)?

Що таке дескриптор та для чого він використовується? Для чого використовують оператор `gcnew`?

29) Опишіть функцію для округлення числових значень `Math::Round` та її параметри.

30) Опишіть функцію для округлення числових значень `Math::Ceiling` та її параметри.

31) Опишіть функцію для округлення числових значень `Math::Floor` та її параметри.

32) Для чого використовуються методи `Math::Truncate` та `Math::Sign`?

33) Для чого потрібен метод `ToString`? Охарактеризуйте описувачі формату даних "C", "D", "E".

34) Опишіть призначення специфікаторів формату даних "F", "G", "N", "P", "R", та "X" і наведіть приклади їх використання.

35) Для чого використовується метод `Parse()`? Опишіть його параметри та наведіть приклади використання.

36) Опишіть призначення та наведіть приклади використання методу `IsNullOrEmpty()` та `IsNullOrWhiteSpace()`.

37) Охарактеризуйте клас `MessageBox` та варіанти створення діалогових вікон з різним набором параметрів. Наведіть приклади.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Базова

1) Хортон, Айвор. Visual C++ 2010: полный курс.: Пер. с англ.- М.: ООО «И.Д. Вильямс», 2011. – 1216 с.
2) Зиборов В.В. MS Visual C ++2010 в среде .NET. Библиотека программиста – СПб.: Питер, 2012. – 320 с.
3) Пахомов Б.И. C/C++ и MS Visual C++ 2010 для начинающих. – СПб.: БХВ-Петербург, 2011. – 736 с.
4) Стенли Б. Липпман, Жози Лажойе, Барбара Э. Му Язык программирования C++. Базовый курс. М.: Вильямс, 2014. – 1120 с.
5) Страуструп, Бьярне. Программирование: принципы и практика использования C++, 2-е изд. - М.: ООО «И.Д. Вильямс», 2016. – 1328 с.
6) Прата С. Язык программирования C++. Лекции и упражнения. Учебник. - СПб.: ООО «ДиаСофтЮП», 2005. - 1104 с.
7) ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ - 3. ПРОЕКТУВАННЯ ПРОГРАМНИХ ДОДАНКІВ: МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ КОМП'ЮТЕРНИХ ПРАКТИКУМІВ для студентів напряму підготовки 151 – «Автоматизація та комп'ютерно-інтегровані технології» [Електронний ресурс] / [уклад. Бендюг В. І., Комариста Б. М.]. – К: 2016. – 255 с.

Допоміжна

8) Мейерс С. Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов. - М.: Питер-ДМК, 2006. – 240 с.
9) Аверкин В.П., Бобровский А.И. и др. под ред. Хомоненко А.Д. Программирование на C++. Учебное пособие.Корона-Принт, 1999. – 252 с.
10) Александреску, Андрей. Современное проектирование на C++. Серия C++ In-Depth, т.3.: Пер. с англ. – М.: Вильямс, 2002. – 336 с.
11) Астахова, И.Ф., Власов, С.В. Язык C++. Учебное пособие. И.Ф. Астахова, С.В. Власов, В.В. Фертиков, А.В. Ларин. – Мн.: Новое знание, 2003. – 203 с.
12) Седжвик, Роберт. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск: Пер. с англ./ Роберт Седжвик. – К.: ДиаСофт, 2001. – 688 с.
13) Седжвик, Роберт. Фундаментальные алгоритмы на C++. Алгоритмы на графах: Пер. с англ./ Роберт Седжвик. – СПб.: ДиаСофтЮП, 2002. – 496 с.

14) Дейтел Х.М., Дейтел П.Дж. Как программировать на C++: Пятое издание. Пер. с англ. – М.: ООО «Бином-Пресс», 2008. - 1456 с.

Інформаційні ресурси

15) Язык программирования C++ [Електрон. ресурс] // Основы программирования на языках Си и C++ для начинающих - Режим доступу: http://cppstudio.com/cat/274/

16) Бендюг Владислав Іванович [Електрон. ресурс] // Офіційний сайт – Режим доступу: http://бендюг.укр/

17) Єдине інформаційне середовище - Національний технічний університет України «КПІ ім. І.Сікорського» [Електрон. ресурс] // Електронний КАМПУС НТУУ «КПІ» – Режим доступу: http://login.kpi.ua/

ДОДАТОК А ТИТУЛЬНИЙ АРКУШ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Хіміко-технологічний факультет

Кафедра кібернетики хіміко-технологічних процесів

Комп’ютерний практикум №1

з дисципліни

«ПРОЕКТУВАННЯ ПРОГРАМНИХ ДОДАТКІВ»

тема: Робота з масивами в консольному додатку CLR

Виконав:
ст. групи ХА-41
Іваненко І.І.

Перевірив:
доц. каф. КХТП
Бендюг В.І.

Оцінка, балів:
Виконання _____ ;
оформлення звіту _____ ;
захист результатів _____.

Загальна оцінка _____.

_____
(підпис викладача)

Київ – 2016

ДОДАТОК Б ІНДИВІДУАЛЬНІ ЗАВДАННЯ ДО РОЗРАХУНКОВО-ГРАФІЧНОЇ РОБОТИ

1. Обчислення кореня заданого нелінійного рівняння $f(x) = 0$ на попередньо визначеному інтервалі $[a, b]^1$ **методом хорд** із заданою точністю ε.
2. Обчислення кореня заданого нелінійного рівняння $f(x) = 0$ на попередньо визначеному інтервалі $[a, b]^1$ **методом дотичних** із заданою точністю ε.
3. Обчислення кореня заданого нелінійного рівняння $f(x) = 0$ на попередньо визначеному інтервалі $[a, b]^1$ **комбінованим методом хорд-дотичних** із заданою точністю ε.
4. Розв'язання системи лінійних алгебраїчних рівнянь **методом ітерацій** із заданою точністю ε.
5. Розв'язання системи лінійних алгебраїчних рівнянь **методом Гауса-Зейделя** із заданою точністю ε.
6. Розв'язання системи двох нелінійних рівнянь **методом ітерацій** із заданою точністю ε.
7. Розв'язання системи двох нелінійних рівнянь **методом Ньютона** із заданою точністю ε.
8. Розв'язання задачі інтерполяції (як прямої, так і оберненої) **методом Лагранжа**.
9. Розв'язання прямої задачі інтерполяції (екстраполяції) **методом Ньютона**.
10. Розв'язання оберненої задачі інтерполяції (екстраполяції) **методом Ньютона**.
11. Розв'язання звичайного диференціального рівняння першого порядку **методом Ейлера**.
12. Розв'язання звичайного диференціального рівняння першого порядку **удосконаленим методом Ейлера**.
13. Розв'язання звичайного диференціального рівняння першого порядку **методом Ейлера-Коші**.
14. Розв'язання звичайного диференціального рівняння першого порядку **методом Ейлера-Коші з наступною ітераційною обробкою**.
15. Розв'язання звичайного диференціального рівняння першого порядку **методом Рунге-Кутта.**
16. Обчислення визначеного інтегралу **методом трапецій**.
17. Обчислення визначеного інтегралу **методом Сімпсона**.

18. Розв’язання системи лінійних алгебраїчних рівнянь **методом ітерацій** із заданою точністю ε.
19. Розв’язання системи лінійних алгебраїчних рівнянь **методом Гауса-Зейделя** із заданою точністю ε.
20. Розв’язання системи лінійних алгебраїчних рівнянь 2-го порядку **за правилом Крамера**.
21. Розв’язання системи лінійних алгебраїчних рівнянь 3-го порядку **за правилом Крамера**.
22. Розв’язання задачі екстраполяції (як прямої, так і оберненої) **методом Лагранжа**.

ДОДАТОК В КОД ПРОГРАМНОГО МОДУЛЯ

В.1 Файл MyForm.cpp

```
#include "MyForm.h"

using namespace Windows_Classes; //ім’я вашого проекту

[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);
	Application::Run(gcnew MyForm());
	return 0;
}
```

В.2 Файл заголовку `MyMethod.h` з визначенням класу

```
#pragma once
//Створений клас для знаходження коренів рівняння методом Золотого
перетину
#include <cmath>
using namespace System;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::Windows::Forms::DataVisualization::Charting;

public ref class MyClass
{
public: //Інтерфейс класу
      MyClass() {};//Конструктор класу за замовчуванням
      MyClass(const float &k1, const float &k2, const float &k3,
            const float &k4, const float &k5, const float &a1,
            const float &b1, const float &e1, const bool &max1) :
            A(a1), B(b1), E(e1), K1(k1), K2(k2), K3(k3), K4(k4),
            K5(k5), MINM(max1) {}; //Конструктор класу
      MyClass(const float &k1, const float &k2, const float &k3,
            const float &a1, const float &b1, const bool &max1) :
            A(a1), B(b1), K1(k1), K2(k2), K3(k3),
            MINM(max1) {}; //Конструктор класу
      //Об'явлення функцій-членів класу без параметрів:
      String^ Method();
      DataTable^ ТаблицяЗначень();
      DataTable^ ТаблицяРозрахунків();
      String^ ТекстовийЗвіт();
      ~MyClass() {};//Об'явлення деструктора класу
private: //Реалізація класу
      //Ініціалізація даних-членів класу:
      double K1 = 15;            //Коефіцієнт функції
      double K2 = 5;             //Коефіцієнт функції
      double K3 = 2;             //Коефіцієнт функції
      double K4 = 0;             //Коефіцієнт функції
      double K5 = 0;             //Коефіцієнт функції
      double A = -100;           //Ліва межа пошуку екстремуму
      double B = 100;            //Права межа пошуку екстремуму
      double E = 0.001;                 //Точність пошуку екстремуму
      bool MINM = true;                 //Шукаємо мінімум чи максимум
      double extrX, extrY;              //Значення екстремуму функції
      //Об'явлення функції-члену класу з параметром –
      //константним посиланням:
      double MyFunc(const double);
      //Об'явлення та створення об'єкта "ТаблицяІтерації"
      //класу DataTable:
      DataTable^ ТаблицяІтерації = gcnew DataTable();
```

```
        //Об'явлення та створення об'єкта "ТаблицяГрафік" класу DataTable:
        DataTable^ ТаблицяГрафік = gcnew DataTable();
        //Об'явлення та створення об'єкта "НабірДаних" класу DataSet:
        DataSet^ НабірДаних = gcnew DataSet();
        String^ Звіт; // Створення об'єкту для збереження текстового звіту
};

String^ MyClass::Method() //Визначення функції-члену класу без параметрів
{
        double y1, y2, x1, x2;
        const double fi = (1 + sqrt(5)) / 2; //Золотий перетин
        unsigned i = 0;
        //Ініціалізація змінних a та b значеннями
        //змінних A та B відповідно:
        double a(A), b(B);
        String^ minmax;
        String^ Результат;
        MINM ? minmax = "мінімум" : minmax = "максимум";
        DataRow^ Ряд = ТаблицяІтерації->NewRow();
        //Додаємо в таблицю стовпчик з іменем "Ітерація"
        //та вказуємо тип його значень:
        ТаблицяІтерації->Columns->Add("Ітерація", int::typeid);
        //Додаємо в таблицю стовпчик з іменем "Значення x1"
        //та вказуємо тип його значень:
        ТаблицяІтерації->Columns->Add("Значення x1", double::typeid);
        //Додаємо в таблицю стовпчик "Значення функції y(x1)"
        //та вказуємо тип його значень:
        ТаблицяІтерації->Columns->Add("Значення функції y(x1)",
                double::typeid);
        //Додаємо в таблицю стовпчик з іменем "Значення x2"
        //та вказуємо тип його значень:
        ТаблицяІтерації->Columns->Add("Значення x2", double::typeid);
        //Додаємо в таблицю стовпчик "Значення функції y(x2)"
        //та вказуємо тип його значень:
        ТаблицяІтерації->Columns->Add("Значення функції y(x2)",
                double::typeid);
        //Задаємо ім'я таблиці даних:
        ТаблицяІтерації->TableName = "Знаходження " + minmax
                + "у функції методом золотого перетину";
        //Задаємо заголовок звіту:
        Звіт += "Знаходження " + minmax + "у функції методом
                золотого перетину\n";
        x2 = a + (b - a) / fi;
        x1 = b - (b - a) / fi;
        y1 = MyFunc(x1);
        y2 = MyFunc(x2);
        do
        {
                y1 = MyFunc(x1);
```

```
            y2 = MyFunc(x2);
            //Перевіряємо що шукаємо мін чи макс:
            if ((!MINM && y1 <= y2) || (MINM && y1 >= y2))
            {
                a = x1;
                x1 = x2;
                x2 = a + (b - a) / fi;
            }
            else
            {
                b = x2;
                x2 = x1;
                x1 = b - (b - a) / fi;
            }
            i++;
            //Формуємо рядок звіту з форматованим виведенням значень:
            Звіт += "Ітерація " + Convert::ToString(i) + ", x1="
                + x1.ToString("F3") + ", y(x1)=" + y1.ToString("F3")
                + ", x2=" + x2.ToString("F3")
                + ", y(x2)=" + y2.ToString("F3") + "\n";
            //Формуємо рядок таблиці з округленням значень:
            Ряд["Ітерація"] = i;
            Ряд["Значення x1"] = Math::Round(x1, 3,
                ::MidpointRounding::AwayFromZero);
            Ряд["Значення функції y(x1)"]
                = Math::Round(y1, 3, ::MidpointRounding::AwayFromZero);
            Ряд["Значення x2"] = Math::Round(x2, 3,
                ::MidpointRounding::AwayFromZero);
            Ряд["Значення функції y(x2)"]
                = Math::Round(y2, 3, ::MidpointRounding::AwayFromZero);
            //Додаємо сформований рядок до таблиці
            ТаблицяІтерації->Rows->Add(Ряд);
            //Створюємо наступний пустий рядок в таблиці
            Ряд = ТаблицяІтерації->NewRow();
        } while (abs(b - a) >= E); //Перевірка досягнення точності
        extrX = (a + b) / 2; //Значення x в точці екстремума
        extrY = MyFunc(extrX); //Значення функції в точці екстремума
        Результат = "Знайдений " + minmax + " функції при x="
            + extrX.ToString("F3") + " y(x)=" + extrY.ToString("F3");
        Звіт += Результат;
        //Додати об'єкт ТаблицяІтерації в DataSet:
        НаборДаних->Tables->Add(ТаблицяІтерації);
        return Результат;
}

double MyClass::MyFunc(const double x)
{//Визначення функції-члену класу з параметром - константним посиланням
        double y;
        y = K1 + K2*pow(x, K3) + K4*pow(x, K5);
```

```
        return y;
}

DataTable^ MyClass::ТаблицяЗначень()
{
        DataRow^ Ряд = ТаблицяГрафік->NewRow();
        double x(A), y;
        //Додаємо в таблицю стовпчик з іменем "Значення x"
        //та вказуємо тип його значень:
        ТаблицяГрафік->Columns->Add("Значення x", double::typeid);
        //Додаємо в таблицю стовпчик "Значення функції y(x)"
        //та вказуємо тип його значень:
        ТаблицяГрафік->Columns->Add("Значення функції y(x)",
                double::typeid);
        do
        {
                y = MyFunc(x);
                //Формуємо рядок таблиці з округленням значень
                //до 3 знаків після коми:
                Ряд["Значення x"] = Math::Round(x, 3,
                        ::MidpointRounding::AwayFromZero);
                Ряд["Значення функції y(x)"]
                        = Math::Round(y, 3, ::MidpointRounding::AwayFromZero);
                //Додаємо сформований рядок до таблиці
                ТаблицяГрафік->Rows->Add(Ряд);
                //Створюємо наступний пустий рядок в таблиці
                Ряд = ТаблицяГрафік->NewRow();
                ++x;
        } while (x <= B);
        НабірДаних->Tables->Add(ТаблицяГрафік);
        return ТаблицяГрафік;
}

String^ MyClass::ТекстовийЗвіт()
{
        return Звіт;
}

DataTable^ MyClass::ТаблицяРозрахунків()
{
        return ТаблицяІтерації;
}
```

В.3 Файл заголовку **MyForm.h** з кодом програмного модуля

```
#pragma once
//Використання об'єктів TabControl та TabPage
//Відкриття та збереження файлів за допомогою діалогових вікон
OpenFileDialog та SaveFileDialog в об'єктах PictureBox та RichTextBox
//Робота з наборами даних DataTable та DataSet, а також таблицею
DataGridView
//Створення та використання власного класу для знаходження екстремуму
функції
//Створення та використання головного та контекстних меню і панелі
інструментів
#include "MyMethod.h"
#include <cmath>

namespace Windows_Classes {
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	//Підключення простору імен для методу Path::GetFileName():
	using namespace System::IO;
	using namespace System::Resources;

	/// <summary>
	/// Сводка для MyForm
	/// </summary>
	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:
		MyForm(void)
		{
			InitializeComponent();
			//
			//TODO: добавьте код конструктора
			//
		}

	protected:
		/// <summary>
		/// Освободить все используемые ресурсы.
		/// </summary>
		~MyForm()
		{
			if (components)
			{
				delete components;
			}
```

```
            }

        private: System::Windows::Forms::MenuStrip^  menuStripГоловнеМеню;
        private: System::Windows::Forms::ToolStripMenuItem^
    файлToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
    відкритиФайлToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
    зберегтиФайлToolStripMenuItem;
        private: System::Windows::Forms::ToolStripSeparator^
    toolStripMenuItem1;
        private: System::Windows::Forms::ToolStripMenuItem^
    вихідToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
    розрахунокToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
              тестовийПрикладToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
    розрахуватиToolStripMenuItem;
        private: System::Windows::Forms::ToolStripSeparator^
    toolStripMenuItem2;
        private: System::Windows::Forms::ToolStripMenuItem^
    очиститиToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
    довідкаToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
    проПрограмуToolStripMenuItem;
        private: System::Windows::Forms::ContextMenuStrip^
    contextMenuStripВихідніДані;
        private: System::Windows::Forms::ContextMenuStrip^
    contextMenuStripРезультати;
        private: System::Windows::Forms::ToolStripMenuItem^
    відкритиФайлToolStripMenuItem1;
        private: System::Windows::Forms::ToolStripMenuItem^
    зберегтиФайлToolStripMenuItem1;
        private: System::Windows::Forms::ToolStripSeparator^
    toolStripMenuItem3;
        private: System::Windows::Forms::ToolStripMenuItem^
    виділитиВсеToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
    скопіюватиToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
    вирізатиToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
    видалитиToolStripMenuItem;
        private: System::Windows::Forms::ContextMenuStrip^
    contextMenuStripГрафік;
        private: System::Windows::Forms::ToolStripMenuItem^
              відкритиГрафічнийФайлToolStripMenuItem;
```

```
        private: System::Windows::Forms::ToolStripMenuItem^
              зберегтиГрафічнийФайлToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
              тестовийПрикладToolStripMenuItem1;
        private: System::Windows::Forms::ToolStripMenuItem^
розрахуватиToolStripMenuItem1;
        private: System::Windows::Forms::ToolStripSeparator^
toolStripMenuItem4;
        private: System::Windows::Forms::ToolStripMenuItem^
очиститиToolStripMenuItem1;
        private: System::Windows::Forms::ToolStrip^
toolStripПанельІнструментів;
        private: System::Windows::Forms::ToolStripSeparator^
toolStripMenuItem5;
        private: System::Windows::Forms::ToolStripMenuItem^
вставитиToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
відмінитиToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
графікToolStripMenuItem;
        private: System::ComponentModel::IContainer^  components;
        private: System::Windows::Forms::ToolStripMenuItem^
звітToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem^
              таблицяРозрахункуToolStripMenuItem;
        private: System::Windows::Forms::ToolStripSeparator^
toolStripMenuItem6;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonВідкрити;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonЗберегти;
        private: System::Windows::Forms::ToolStripSeparator^
toolStripSeparator1;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonТест;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonРозрахувати;
        private: System::Windows::Forms::ToolStripSeparator^
toolStripSeparator2;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonГрафік;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonЗвіт;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonТаблиця;
        private: System::Windows::Forms::ToolStripSeparator^
toolStripSeparator3;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonОчистити;
```

```
        private: System::Windows::Forms::ToolStripSeparator^
toolStripSeparator4;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonПроПрограму;
        private: System::Windows::Forms::ToolStripSeparator^
toolStripSeparator5;
        private: System::Windows::Forms::ToolStripButton^
toolStripButtonВихід;
        private: System::Windows::Forms::Panel^  panelГоловна;
        private: System::Windows::Forms::TabControl^  tabControlСторінки;
        private: System::Windows::Forms::TabPage^  tabPageДані;
        private: System::Windows::Forms::Panel^  panelКоефіцієнтиТочність;
        private: System::Windows::Forms::GroupBox^  groupBoxКоефіцієнти;
        private: System::Windows::Forms::TextBox^  textBox_k5;
        private: System::Windows::Forms::TextBox^  textBox_k4;
        private: System::Windows::Forms::Label^  label_k4;
        private: System::Windows::Forms::TextBox^  textBox_k3;
        private: System::Windows::Forms::TextBox^  textBox_k2;
        private: System::Windows::Forms::Label^  label_k2;
        private: System::Windows::Forms::TextBox^  textBox_k1;
        private: System::Windows::Forms::Label^  label_k1;
        private: System::Windows::Forms::Label^  label_k3;
        private: System::Windows::Forms::GroupBox^  groupBoxТочність;
        private: System::Windows::Forms::TextBox^  textBox_e;
        private: System::Windows::Forms::Label^  label_e;
        private: System::Windows::Forms::Panel^  panelЕкстремумІнтервал;
        private: System::Windows::Forms::GroupBox^  groupBoxІнтервалПошуку;
        private: System::Windows::Forms::TextBox^  textBox_b;
        private: System::Windows::Forms::Label^  label_b;
        private: System::Windows::Forms::TextBox^  textBox_a;
        private: System::Windows::Forms::Label^  label_a;
        private: System::Windows::Forms::GroupBox^  groupBoxЕкстремум;
        private: System::Windows::Forms::CheckedListBox^
checkedListBoxЕкстремум;
        private: System::Windows::Forms::TabPage^  tabPageРезультат;
        private: System::Windows::Forms::RichTextBox^  richTextBoxЗвіт;
        private: System::Windows::Forms::TabPage^  tabPageГрафік;
        private:
System::Windows::Forms::DataVisualization::Charting::Chart^  chartГрафік;
        private: System::Windows::Forms::PictureBox^  pictureBox1;
        private: System::Windows::Forms::TabPage^  tabPageІтерації;
        private: System::Windows::Forms::DataGridView^
dataGridViewІтерації;

        private: MyClass^
Екстремум;//////////////////////////////////////////////////////

        private:
            /// <summary>
```

```
            /// Обязательная переменная конструктора.
            /// </summary>

#pragma region Windows Form Designer generated code
            /// <summary>
            /// Требуемый метод для поддержки конструктора — не изменяйте
            /// содержимое этого метода с помощью редактора кода.
            /// </summary>
      void InitializeComponent(void)
      {
            this->components = (gcnew
System::ComponentModel::Container());
            System::ComponentModel::ComponentResourceManager^  resources =
(gcnew

      System::ComponentModel::ComponentResourceManager(MyForm::typeid));

      System::Windows::Forms::DataVisualization::Charting::ChartArea^
chartArea1 =
                  (gcnew System::Windows::Forms::DataVisualization
                  ::Charting::ChartArea());
            System::Windows::Forms::DataVisualization::Charting::Legend^
legend1 = (gcnew

      System::Windows::Forms::DataVisualization::Charting::Legend());
            System::Windows::Forms::DataVisualization::Charting::Series^
series1 = (gcnew

      System::Windows::Forms::DataVisualization::Charting::Series());
            this->contextMenuStripВихідніДані = (gcnew
                  System::Windows::Forms::ContextMenuStrip(this-
>components));
            this->тестовийПрикладToolStripMenuItem1 = (gcnew
                  System::Windows::Forms::ToolStripMenuItem());
            this->розрахуватиToolStripMenuItem1 = (gcnew
                  System::Windows::Forms::ToolStripMenuItem());
            this->toolStripMenuItem4 = (gcnew
                  System::Windows::Forms::ToolStripSeparator());
            this->очиститиToolStripMenuItem1 = (gcnew
                  System::Windows::Forms::ToolStripMenuItem());
            this->contextMenuStripРезультати = (gcnew
                  System::Windows::Forms::ContextMenuStrip(this-
>components));
            this->відкритиФайлToolStripMenuItem1 = (gcnew
                  System::Windows::Forms::ToolStripMenuItem());
            this->зберегтиФайлToolStripMenuItem1 = (gcnew
                  System::Windows::Forms::ToolStripMenuItem());
            this->toolStripMenuItem3 = (gcnew
                  System::Windows::Forms::ToolStripSeparator());
```

```
            this->виділитиВсеToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->скопіюватиToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->вирізатиToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->вставитиToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->toolStripMenuItem5 = (gcnew
                System::Windows::Forms::ToolStripSeparator());
            this->видалитиToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->відмінитиToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->contextMenuStripГрафік = (gcnew
                System::Windows::Forms::ContextMenuStrip(this-
>components));
            this->відкритиГрафічнийФайлToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->зберегтиГрафічнийФайлToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->menuStripГоловнеМеню = (gcnew
System::Windows::Forms::MenuStrip());
            this->файлToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->відкритиФайлToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->зберегтиФайлToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->toolStripMenuItem1 = (gcnew
                System::Windows::Forms::ToolStripSeparator());
            this->вихідToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->розрахунокToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->тестовийПрикладToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->розрахуватиToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->toolStripMenuItem2 = (gcnew
                System::Windows::Forms::ToolStripSeparator());
            this->графікToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->звітToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->таблицяРозрахункуToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->toolStripMenuItem6 = (gcnew
                System::Windows::Forms::ToolStripSeparator());
```

```
            this->очиститиToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->довідкаToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->проПрограмуToolStripMenuItem = (gcnew
                System::Windows::Forms::ToolStripMenuItem());
            this->toolStripПанельІнструментів = (gcnew
                System::Windows::Forms::ToolStrip());
            this->toolStripButtonВідкрити = (gcnew
                System::Windows::Forms::ToolStripButton());
            this->toolStripButtonЗберегти = (gcnew
                System::Windows::Forms::ToolStripButton());
            this->toolStripSeparator1 = (gcnew
                System::Windows::Forms::ToolStripSeparator());
            this->toolStripButtonТест = (gcnew
System::Windows::Forms::ToolStripButton());
            this->toolStripButtonРозрахувати = (gcnew
                System::Windows::Forms::ToolStripButton());
            this->toolStripSeparator2 = (gcnew
                System::Windows::Forms::ToolStripSeparator());
            this->toolStripButtonГрафік = (gcnew
                System::Windows::Forms::ToolStripButton());
            this->toolStripButtonЗвіт = (gcnew
System::Windows::Forms::ToolStripButton());
            this->toolStripButtonТаблиця = (gcnew
                System::Windows::Forms::ToolStripButton());
            this->toolStripSeparator3 = (gcnew
                System::Windows::Forms::ToolStripSeparator());
            this->toolStripButtonОчистити = (gcnew
                System::Windows::Forms::ToolStripButton());
            this->toolStripSeparator4 = (gcnew
                System::Windows::Forms::ToolStripSeparator());
            this->toolStripButtonПроПрограму = (gcnew
                System::Windows::Forms::ToolStripButton());
            this->toolStripSeparator5 = (gcnew
                System::Windows::Forms::ToolStripSeparator());
            this->toolStripButtonВихід = (gcnew
                System::Windows::Forms::ToolStripButton());
            this->panelГоловна = (gcnew System::Windows::Forms::Panel());
            this->tabControlСторінки = (gcnew
System::Windows::Forms::TabControl());
            this->tabPageДані = (gcnew System::Windows::Forms::TabPage());
            this->panelКоефіцієнтиТочність = (gcnew
System::Windows::Forms::Panel());
            this->groupBoxКоефіцієнти = (gcnew
System::Windows::Forms::GroupBox());
            this->textBox_k5 = (gcnew System::Windows::Forms::TextBox());
            this->textBox_k4 = (gcnew System::Windows::Forms::TextBox());
            this->label_k4 = (gcnew System::Windows::Forms::Label());
```

```
            this->textBox_k3 = (gcnew System::Windows::Forms::TextBox());
            this->textBox_k2 = (gcnew System::Windows::Forms::TextBox());
            this->label_k2 = (gcnew System::Windows::Forms::Label());
            this->textBox_k1 = (gcnew System::Windows::Forms::TextBox());
            this->label_k1 = (gcnew System::Windows::Forms::Label());
            this->label_k3 = (gcnew System::Windows::Forms::Label());
            this->groupBoxТочність = (gcnew
System::Windows::Forms::GroupBox());
            this->textBox_e = (gcnew System::Windows::Forms::TextBox());
            this->label_e = (gcnew System::Windows::Forms::Label());
            this->panelЕкстремумІнтервал = (gcnew
System::Windows::Forms::Panel());
            this->groupBoxІнтервалПошуку = (gcnew
System::Windows::Forms::GroupBox());
            this->textBox_b = (gcnew System::Windows::Forms::TextBox());
            this->label_b = (gcnew System::Windows::Forms::Label());
            this->textBox_a = (gcnew System::Windows::Forms::TextBox());
            this->label_a = (gcnew System::Windows::Forms::Label());
            this->groupBoxЕкстремум = (gcnew
System::Windows::Forms::GroupBox());
            this->checkedListBoxЕкстремум = (gcnew
                  System::Windows::Forms::CheckedListBox());
            this->tabPageРезультат = (gcnew
System::Windows::Forms::TabPage());
            this->richTextBoxЗвіт = (gcnew
System::Windows::Forms::RichTextBox());
            this->tabPageГрафік = (gcnew
System::Windows::Forms::TabPage());
            this->chartГрафік = (gcnew

      System::Windows::Forms::DataVisualization::Charting::Chart());
            this->pictureBox1 = (gcnew
System::Windows::Forms::PictureBox());
            this->tabPageІтерації = (gcnew
System::Windows::Forms::TabPage());
            this->dataGridViewІтерації = (gcnew
System::Windows::Forms::DataGridView());
            this->contextMenuStripВихідніДані->SuspendLayout();
            this->contextMenuStripРезультати->SuspendLayout();
            this->contextMenuStripГрафік->SuspendLayout();
            this->menuStripГоловнеМеню->SuspendLayout();
            this->toolStripПанельІнструментів->SuspendLayout();
            this->panelГоловна->SuspendLayout();
            this->tabControlСторінки->SuspendLayout();
            this->tabPageДані->SuspendLayout();
            this->panelКоефіцієнтиТочність->SuspendLayout();
            this->groupBoxКоефіцієнти->SuspendLayout();
            this->groupBoxТочність->SuspendLayout();
            this->panelЕкстремумІнтервал->SuspendLayout();
```

```
            this->groupBoxІнтервалПошуку->SuspendLayout();
            this->groupBoxЕкстремум->SuspendLayout();
            this->tabPageРезультат->SuspendLayout();
            this->tabPageГрафік->SuspendLayout();

      (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
                  ->chartГрафік))->BeginInit();

      (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
                  ->pictureBox1))->BeginInit();
            this->tabPageІтерації->SuspendLayout();

      (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
                  ->dataGridViewІтерації))->BeginInit();
            this->SuspendLayout();
            //
            // contextMenuStripВихідніДані
            //
            this->contextMenuStripВихідніДані->Items->AddRange(gcnew
      cli::array< System
                  ::Windows::Forms::ToolStripItem^  >(4)
            {
                  this->тестовийПрикладToolStripMenuItem1,
                  this->розрахуватиToolStripMenuItem1,
                  this->toolStripMenuItem4,
                  this->очиститиToolStripMenuItem1
            });
            this->contextMenuStripВихідніДані->Name =
L"contextMenuStripВихідніДані";
            this->contextMenuStripВихідніДані->Size =
System::Drawing::Size(212, 76);
            //
            // тестовийПрикладToolStripMenuItem1
            //
            this->тестовийПрикладToolStripMenuItem1->Image
                  = (cli::safe_cast<System::Drawing::Image^>(resources
                  -
            >GetObject(L"тестовийПрикладToolStripMenuItem1.Image")));
            this->тестовийПрикладToolStripMenuItem1->Name
                  = L"тестовийПрикладToolStripMenuItem1";
            this->тестовийПрикладToolStripMenuItem1->ShortcutKeys
                  =
            static_cast<System::Windows::Forms::Keys>((System::Windows
                  ::Forms::Keys::Alt | System::Windows::Forms::Keys::T));
            this->тестовийПрикладToolStripMenuItem1->Size =
System::Drawing
                  ::Size(211, 22);
            this->тестовийПрикладToolStripMenuItem1->Text = L"Тестовий
приклад";
```

```
            this->тестовийПрикладToolStripMenuItem1->Click += gcnew System
                ::EventHandler(this,
            &MyForm::тестовийПрикладToolStripMenuItem_Click);
            //
            // розрахуватиToolStripMenuItem1
            //
            this->розрахуватиToolStripMenuItem1->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"розрахуватиToolStripMenuItem1.Image")));
            this->розрахуватиToolStripMenuItem1->Name =
L"розрахуватиToolStripMenuItem1";
            this->розрахуватиToolStripMenuItem1->ShortcutKeys =
static_cast
                <System::Windows::Forms::Keys>((System::Windows::Forms::
            Keys
                ::Alt | System::Windows::Forms::Keys::C));
            this->розрахуватиToolStripMenuItem1->Size =
System::Drawing::Size(211, 22);
            this->розрахуватиToolStripMenuItem1->Text = L"Розрахувати";
            this->розрахуватиToolStripMenuItem1->Click += gcnew System
                ::EventHandler(this,
            &MyForm::розрахуватиToolStripMenuItem_Click);
            //
            // toolStripMenuItem4
            //
            this->toolStripMenuItem4->Name = L"toolStripMenuItem4";
            this->toolStripMenuItem4->Size = System::Drawing::Size(208,
6);
            //
            // очиститиToolStripMenuItem1
            //
            this->очиститиToolStripMenuItem1->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"очиститиToolStripMenuItem1.Image")));
            this->очиститиToolStripMenuItem1->Name =
L"очиститиToolStripMenuItem1";
            this->очиститиToolStripMenuItem1->ShortcutKeys =
static_cast<System
                ::Windows::Forms::Keys>((System::Windows::Forms::Keys::A
            lt |
                System::Windows::Forms::Keys::Delete));
            this->очиститиToolStripMenuItem1->Size =
System::Drawing::Size(211, 22);
            this->очиститиToolStripMenuItem1->Text = L"Очистити";
            this->очиститиToolStripMenuItem1->Click += gcnew
System::EventHandler(this,
                &MyForm::очиститиToolStripMenuItem_Click);
```

```
            //
            // contextMenuStripРезультати
            //
            this->contextMenuStripРезультати->Items->AddRange(gcnew
                cli::array< System::Windows::Forms::ToolStripItem^
            >(10)
            {
                this->відкритиФайлToolStripMenuItem1,
                this->зберегтиФайлToolStripMenuItem1,
                this->toolStripMenuItem3,
                this->виділитиВсеToolStripMenuItem,
                this->скопіюватиToolStripMenuItem,
                this->вирізатиToolStripMenuItem,
                this->вставитиToolStripMenuItem,
                this->toolStripMenuItem5,
                this->видалитиToolStripMenuItem,
                this->відмінитиToolStripMenuItem
            });
            this->contextMenuStripРезультати->Name =
L"contextMenuStripРезультати";
            this->contextMenuStripРезультати->Size =
System::Drawing::Size(220, 192);
            //
            // відкритиФайлToolStripMenuItem1
            //
            this->відкритиФайлToolStripMenuItem1->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"відкритиФайлToolStripMenuItem1.Image")));
            this->відкритиФайлToolStripMenuItem1->Name
                = L"відкритиФайлToolStripMenuItem1";
            this->відкритиФайлToolStripMenuItem1->ShortcutKeys =
static_cast<System
                ::Windows::Forms::Keys>((System::Windows::Forms::Keys::C
            ontrol |
                System::Windows::Forms::Keys::O));
            this->відкритиФайлToolStripMenuItem1->Size =
System::Drawing::Size(219, 22);
            this->відкритиФайлToolStripMenuItem1->Text = L"Відкрити файл
RTF";
            this->відкритиФайлToolStripMenuItem1->Click += gcnew System
                ::EventHandler(this,
            &MyForm::відкритиФайлToolStripMenuItem_Click);
            //
            // зберегтиФайлToolStripMenuItem1
            //
            this->зберегтиФайлToolStripMenuItem1->Image =
(cli::safe_cast<System::Drawing
```

```
            ::Image^>(resources-
        >GetObject(L"зберегтиФайлToolStripMenuItem1.Image")));
        this->зберегтиФайлToolStripMenuItem1->Name
            = L"зберегтиФайлToolStripMenuItem1";
        this->зберегтиФайлToolStripMenuItem1->ShortcutKeys =
static_cast<System
            ::Windows::Forms::Keys>((System::Windows::Forms::Keys::C
        ontrol |
            System::Windows::Forms::Keys::S));
        this->зберегтиФайлToolStripMenuItem1->Size =
System::Drawing::Size(219, 22);
        this->зберегтиФайлToolStripMenuItem1->Text = L"Зберегти файл
RTF";
        this->зберегтиФайлToolStripMenuItem1->Click += gcnew System
            ::EventHandler(this,
        &MyForm::зберегтиФайлToolStripMenuItem_Click);
        //
        // toolStripMenuItem3
        //
        this->toolStripMenuItem3->Name = L"toolStripMenuItem3";
        this->toolStripMenuItem3->Size = System::Drawing::Size(216,
6);
        //
        // виділитиВсеToolStripMenuItem
        //
        this->виділитиВсеToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
            ::Image^>(resources-
        >GetObject(L"виділитиВсеToolStripMenuItem.Image")));
        this->виділитиВсеToolStripMenuItem->Name =
L"виділитиВсеToolStripMenuItem";
        this->виділитиВсеToolStripMenuItem->ShortcutKeys =
static_cast<System::Windows
            ::Forms::Keys>((System::Windows::Forms::Keys::Control |
            System::Windows::Forms::Keys::A));
        this->виділитиВсеToolStripMenuItem->Size =
System::Drawing::Size(219, 22);
        this->виділитиВсеToolStripMenuItem->Text = L"Виділити все";
        this->виділитиВсеToolStripMenuItem->Click += gcnew System
            ::EventHandler(this,
        &MyForm::виділитиВсеToolStripMenuItem_Click);
        //
        // скопіюватиToolStripMenuItem
        //
        this->скопіюватиToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
            ::Image^>(resources-
        >GetObject(L"скопіюватиToolStripMenuItem.Image")));
```

```
            this->скопіюватиToolStripMenuItem->Name =
L"скопіюватиToolStripMenuItem";
            this->скопіюватиToolStripMenuItem->ShortcutKeys =
static_cast<System::Windows
                ::Forms::Keys>((System::Windows::Forms::Keys::Control |
                System::Windows::Forms::Keys::C));
            this->скопіюватиToolStripMenuItem->Size =
System::Drawing::Size(219, 22);
            this->скопіюватиToolStripMenuItem->Text = L"Скопіювати";
            this->скопіюватиToolStripMenuItem->Click += gcnew System
                ::EventHandler(this,
            &MyForm::скопіюватиToolStripMenuItem_Click);
            //
            // вирізатиToolStripMenuItem
            //
            this->вирізатиToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"вирізатиToolStripMenuItem.Image")));
            this->вирізатиToolStripMenuItem->Name =
L"вирізатиToolStripMenuItem";
            this->вирізатиToolStripMenuItem->ShortcutKeys =
static_cast<System::Windows
                ::Forms::Keys>((System::Windows::Forms::Keys::Control |
                System::Windows::Forms::Keys::X));
            this->вирізатиToolStripMenuItem->Size =
System::Drawing::Size(219, 22);
            this->вирізатиToolStripMenuItem->Text = L"Вирізати";
            this->вирізатиToolStripMenuItem->Click += gcnew System
                ::EventHandler(this,
            &MyForm::вирізатиToolStripMenuItem_Click);
            //
            // вставитиToolStripMenuItem
            //
            this->вставитиToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"вставитиToolStripMenuItem.Image")));
            this->вставитиToolStripMenuItem->Name =
L"вставитиToolStripMenuItem";
            this->вставитиToolStripMenuItem->ShortcutKeys =
static_cast<System::Windows
                ::Forms::Keys>((System::Windows::Forms::Keys::Control |
                System::Windows::Forms::Keys::V));
            this->вставитиToolStripMenuItem->Size =
System::Drawing::Size(219, 22);
            this->вставитиToolStripMenuItem->Text = L"Вставити";
            this->вставитиToolStripMenuItem->Click += gcnew System
```

```
                ::EventHandler(this,
            &MyForm::вставитиToolStripMenuItem_Click);
            //
            // toolStripMenuItem5
            //
            this->toolStripMenuItem5->Name = L"toolStripMenuItem5";
            this->toolStripMenuItem5->Size = System::Drawing::Size(216,
6);
            //
            // видалитиToolStripMenuItem
            //
            this->видалитиToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"видалитиToolStripMenuItem.Image")));
            this->видалитиToolStripMenuItem->Name =
L"видалитиToolStripMenuItem";
            this->видалитиToolStripMenuItem->ShortcutKeys =
static_cast<System::Windows
                ::Forms::Keys>((System::Windows::Forms::Keys::Alt |
                System::Windows::Forms::Keys::Delete));
            this->видалитиToolStripMenuItem->Size =
System::Drawing::Size(219, 22);
            this->видалитиToolStripMenuItem->Text = L"Очистити";
            this->видалитиToolStripMenuItem->Click += gcnew System
                ::EventHandler(this,
            &MyForm::видалитиToolStripMenuItem_Click);
            //
            // відмінитиToolStripMenuItem
            //
            this->відмінитиToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"відмінитиToolStripMenuItem.Image")));
            this->відмінитиToolStripMenuItem->Name =
L"відмінитиToolStripMenuItem";
            this->відмінитиToolStripMenuItem->ShortcutKeys =
static_cast<System::Windows
                ::Forms::Keys>((System::Windows::Forms::Keys::Control |
                System::Windows::Forms::Keys::Z));
            this->відмінитиToolStripMenuItem->Size =
System::Drawing::Size(219, 22);
            this->відмінитиToolStripMenuItem->Text = L"Відмінити";
            this->відмінитиToolStripMenuItem->Click += gcnew System
                ::EventHandler(this,
            &MyForm::відмінитиToolStripMenuItem_Click);
            //
            // contextMenuStripГрафік
            //
```

```
            this->contextMenuStripГрафік->Items->AddRange(gcnew
cli::array< System
                ::Windows::Forms::ToolStripItem^  >(2)
            {
                this->відкритиГрафічнийФайлToolStripMenuItem,
                this->зберегтиГрафічнийФайлToolStripMenuItem
            });
            this->contextMenuStripГрафік->Name =
L"contextMenuStripГрафік";
            this->contextMenuStripГрафік->Size =
System::Drawing::Size(259, 48);
            //
            // відкритиГрафічнийФайлToolStripMenuItem
            //
            this->відкритиГрафічнийФайлToolStripMenuItem->Image =
(cli::safe_cast<System
                ::Drawing::Image^>(resources->GetObject
                (L"відкритиГрафічнийФайлToolStripMenuItem.Image")));
            this->відкритиГрафічнийФайлToolStripMenuItem->Name
                = L"відкритиГрафічнийФайлToolStripMenuItem";
            this->відкритиГрафічнийФайлToolStripMenuItem->ShortcutKeys
                =
            static_cast<System::Windows::Forms::Keys>((System::Windows
                ::Forms::Keys::Control |
            System::Windows::Forms::Keys::O));
            this->відкритиГрафічнийФайлToolStripMenuItem->Size
                = System::Drawing::Size(258, 22);
            this->відкритиГрафічнийФайлToolStripMenuItem->Text
                = L"Відкрити графічний файл";
            this->відкритиГрафічнийФайлToolStripMenuItem->Click += gcnew
System
                ::EventHandler(this,
            &MyForm::відкритиФайлToolStripMenuItem_Click);
            //
            // зберегтиГрафічнийФайлToolStripMenuItem
            //
            this->зберегтиГрафічнийФайлToolStripMenuItem->Image
                = (cli::safe_cast<System::Drawing::Image^>(resources
                -
            >GetObject(L"зберегтиГрафічнийФайлToolStripMenuItem.Image")));
            this->зберегтиГрафічнийФайлToolStripMenuItem->Name
                = L"зберегтиГрафічнийФайлToolStripMenuItem";
            this->зберегтиГрафічнийФайлToolStripMenuItem->ShortcutKeys
                =
            static_cast<System::Windows::Forms::Keys>((System::Windows::Fo
            rms
                ::Keys::Control | System::Windows::Forms::Keys::S));
            this->зберегтиГрафічнийФайлToolStripMenuItem->Size
                = System::Drawing::Size(258, 22);
```

```
            this->зберегтиГрафічнийФайлToolStripMenuItem->Text
                  = L"Зберегти графічний файл";
            this->зберегтиГрафічнийФайлToolStripMenuItem->Click += gcnew
System
                  ::EventHandler(this,
            &MyForm::зберегтиФайлToolStripMenuItem_Click);
            //
            // menuStripГоловнеМеню
            //
            this->menuStripГоловнеМеню->Items->AddRange(gcnew cli::array<
System
                  ::Windows::Forms::ToolStripItem^  >(3)
            {
                  this->файлToolStripMenuItem,
                  this->розрахунокToolStripMenuItem,
                  this->довідкаToolStripMenuItem
            });
            this->menuStripГоловнеМеню->Location =
System::Drawing::Point(0, 0);
            this->menuStripГоловнеМеню->Name = L"menuStripГоловнеМеню";
            this->menuStripГоловнеМеню->Size = System::Drawing::Size(639,
24);
            this->menuStripГоловнеМеню->TabIndex = 3;
            this->menuStripГоловнеМеню->Text = L"menuStrip1";
            //
            // файлToolStripMenuItem
            //
            this->файлToolStripMenuItem->DropDownItems->AddRange(gcnew cli
                  ::array< System::Windows::Forms::ToolStripItem^  >(4)
            {
                  this->відкритиФайлToolStripMenuItem,
                  this->зберегтиФайлToolStripMenuItem,
                  this->toolStripMenuItem1,
                  this->вихідToolStripMenuItem
            });
            this->файлToolStripMenuItem->Name = L"файлToolStripMenuItem";
            this->файлToolStripMenuItem->Size = System::Drawing::Size(48,
20);
            this->файлToolStripMenuItem->Text = L"Файл";
            //
            // відкритиФайлToolStripMenuItem
            //
            this->відкритиФайлToolStripMenuItem->Image
                  = (cli::safe_cast<System::Drawing::Image^>(resources
                  ->GetObject(L"відкритиФайлToolStripMenuItem.Image")));
            this->відкритиФайлToolStripMenuItem->Name =
L"відкритиФайлToolStripMenuItem";
            this->відкритиФайлToolStripMenuItem->ShortcutKeys
```

```
                =
            static_cast<System::Windows::Forms::Keys>((System::Windows
                ::Forms::Keys::Control |
            System::Windows::Forms::Keys::O));
            this->відкритиФайлToolStripMenuItem->Size =
System::Drawing::Size(197, 22);
            this->відкритиФайлToolStripMenuItem->Text = L"Відкрити файл";
            this->відкритиФайлToolStripMenuItem->Click += gcnew System
                ::EventHandler(this,
            &MyForm::відкритиФайлToolStripMenuItem_Click);
            //
            // зберегтиФайлToolStripMenuItem
            //
            this->зберегтиФайлToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"зберегтиФайлToolStripMenuItem.Image")));
            this->зберегтиФайлToolStripMenuItem->Name =
L"зберегтиФайлToolStripMenuItem";
            this->зберегтиФайлToolStripMenuItem->ShortcutKeys =
static_cast<System
                ::Windows::Forms::Keys>((System::Windows::Forms::Keys::C
            ontrol |
                System::Windows::Forms::Keys::S));
            this->зберегтиФайлToolStripMenuItem->Size =
System::Drawing::Size(197, 22);
            this->зберегтиФайлToolStripMenuItem->Text = L"Зберегти файл";
            this->зберегтиФайлToolStripMenuItem->Click += gcnew System
                ::EventHandler(this,
            &MyForm::зберегтиФайлToolStripMenuItem_Click);
            //
            // toolStripMenuItem1
            //
            this->toolStripMenuItem1->Name = L"toolStripMenuItem1";
            this->toolStripMenuItem1->Size = System::Drawing::Size(194,
6);
            //
            // вихідToolStripMenuItem
            //
            this->вихідToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing::Image^>
                (resources-
            >GetObject(L"вихідToolStripMenuItem.Image")));
            this->вихідToolStripMenuItem->Name =
L"вихідToolStripMenuItem";
            this->вихідToolStripMenuItem->ShortcutKeys =
static_cast<System::Windows
                ::Forms::Keys>((System::Windows::Forms::Keys::Alt |
                System::Windows::Forms::Keys::F4));
```

```
            this->вихідToolStripMenuItem->Size =
System::Drawing::Size(197, 22);
            this->вихідToolStripMenuItem->Text = L"Вихід";
            this->вихідToolStripMenuItem->Click += gcnew System
                  ::EventHandler(this,
            &MyForm::вихідToolStripMenuItem_Click);
            //
            // розрахунокToolStripMenuItem
            //
            this->розрахунокToolStripMenuItem->DropDownItems-
>AddRange(gcnew cli
                  ::array< System::Windows::Forms::ToolStripItem^  >(8)
            {
                  this->тестовийПрикладToolStripMenuItem,
                  this->розрахуватиToolStripMenuItem,
                  this->toolStripMenuItem2,
                  this->графікToolStripMenuItem,
                  this->звітToolStripMenuItem,
                  this->таблицяРозрахункуToolStripMenuItem,
                  this->toolStripMenuItem6,
                  this->очиститиToolStripMenuItem
            });
            this->розрахунокToolStripMenuItem->Name =
L"розрахунокToolStripMenuItem";
            this->розрахунокToolStripMenuItem->Size =
System::Drawing::Size(82, 20);
            this->розрахунокToolStripMenuItem->Text = L"Розрахунок";
            //
            // тестовийПрикладToolStripMenuItem
            //
            this->тестовийПрикладToolStripMenuItem->Image =
(cli::safe_cast<System
                  ::Drawing::Image^>(resources->
                  GetObject(L"тестовийПрикладToolStripMenuItem.Image")));
            this->тестовийПрикладToolStripMenuItem->Name
                  = L"тестовийПрикладToolStripMenuItem";
            this->тестовийПрикладToolStripMenuItem->ShortcutKeys =
static_cast<System
                  ::Windows::Forms::Keys>((System::Windows::Forms::Keys::A
            lt |
                  System::Windows::Forms::Keys::T));
            this->тестовийПрикладToolStripMenuItem->Size =
System::Drawing::Size(230, 22);
            this->тестовийПрикладToolStripMenuItem->Text = L"Тестовий
приклад";
            this->тестовийПрикладToolStripMenuItem->Click += gcnew
System::
                  EventHandler(this,
            &MyForm::тестовийПрикладToolStripMenuItem_Click);
```

```
            //
            // розрахуватиToolStripMenuItem
            //
            this->розрахуватиToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"розрахуватиToolStripMenuItem.Image")));
            this->розрахуватиToolStripMenuItem->Name =
L"розрахуватиToolStripMenuItem";
            this->розрахуватиToolStripMenuItem->ShortcutKeys =
static_cast<System
                ::Windows::Forms::Keys>((System::Windows::Forms::Keys::A
            lt |
                System::Windows::Forms::Keys::C));
            this->розрахуватиToolStripMenuItem->Size =
System::Drawing::Size(230, 22);
            this->розрахуватиToolStripMenuItem->Text = L"Розрахувати";
            this->розрахуватиToolStripMenuItem->Click += gcnew System
                ::EventHandler(this,
            &MyForm::розрахуватиToolStripMenuItem_Click);
            //
            // toolStripMenuItem2
            //
            this->toolStripMenuItem2->Name = L"toolStripMenuItem2";
            this->toolStripMenuItem2->Size = System::Drawing::Size(227,
6);
            //
            // графікToolStripMenuItem
            //
            this->графікToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"графікToolStripMenuItem.Image")));
            this->графікToolStripMenuItem->Name =
L"графікToolStripMenuItem";
            this->графікToolStripMenuItem->ShortcutKeys =
static_cast<System::Windows
                ::Forms::Keys>((System::Windows::Forms::Keys::Control |
                System::Windows::Forms::Keys::G));
            this->графікToolStripMenuItem->Size =
System::Drawing::Size(230, 22);
            this->графікToolStripMenuItem->Text = L"Графік";
            this->графікToolStripMenuItem->Click += gcnew System
                ::EventHandler(this,
            &MyForm::графікToolStripMenuItem_Click);
            //
            // звітToolStripMenuItem
            //
```

```
            this->звітToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"звітToolStripMenuItem.Image")));
            this->звітToolStripMenuItem->Name = L"звітToolStripMenuItem";
            this->звітToolStripMenuItem->ShortcutKeys =
static_cast<System::Windows
                ::Forms::Keys>((System::Windows::Forms::Keys::Control |
                System::Windows::Forms::Keys::R));
            this->звітToolStripMenuItem->Size = System::Drawing::Size(230,
22);
            this->звітToolStripMenuItem->Text = L"Звіт";
            this->звітToolStripMenuItem->Click += gcnew System
                ::EventHandler(this,
            &MyForm::звітToolStripMenuItem_Click);
            //
            // таблицяРозрахункуToolStripMenuItem
            //
            this->таблицяРозрахункуToolStripMenuItem->Image =
(cli::safe_cast<System
                ::Drawing::Image^>(resources->GetObject
                (L"таблицяРозрахункуToolStripMenuItem.Image")));
            this->таблицяРозрахункуToolStripMenuItem->Name
                = L"таблицяРозрахункуToolStripMenuItem";
            this->таблицяРозрахункуToolStripMenuItem->ShortcutKeys =
static_cast<System
                ::Windows::Forms::Keys>((System::Windows::Forms::Keys::C
            ontrol |
                System::Windows::Forms::Keys::T));
            this->таблицяРозрахункуToolStripMenuItem->Size
                = System::Drawing::Size(230, 22);
            this->таблицяРозрахункуToolStripMenuItem->Text = L"Таблиця
розрахунків";
            this->таблицяРозрахункуToolStripMenuItem->Click += gcnew
System::
                EventHandler(this,
            &MyForm::таблицяРозрахункуToolStripMenuItem_Click);
            //
            // toolStripMenuItem6
            //
            this->toolStripMenuItem6->Name = L"toolStripMenuItem6";
            this->toolStripMenuItem6->Size = System::Drawing::Size(227,
6);
            //
            // очиститиToolStripMenuItem
            //
            this->очиститиToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing
```

```
            ::Image^>(resources-
        >GetObject(L"очиститиToolStripMenuItem.Image")));
        this->очиститиToolStripMenuItem->Name =
L"очиститиToolStripMenuItem";
        this->очиститиToolStripMenuItem->ShortcutKeys =
static_cast<System
            ::Windows::Forms::Keys>((System::Windows::Forms::Keys::A
        lt |
            System::Windows::Forms::Keys::Delete));
        this->очиститиToolStripMenuItem->Size =
System::Drawing::Size(230, 22);
        this->очиститиToolStripMenuItem->Text = L"Очистити";
        this->очиститиToolStripMenuItem->Click += gcnew System
            ::EventHandler(this,
        &MyForm::очиститиToolStripMenuItem_Click);
        //
        // довідкаToolStripMenuItem
        //
        this->довідкаToolStripMenuItem->DropDownItems->AddRange(gcnew
cli
            ::array< System::Windows::Forms::ToolStripItem^  >(1)
            { this->проПрограмуToolStripMenuItem });
        this->довідкаToolStripMenuItem->Name =
L"довідкаToolStripMenuItem";
        this->довідкаToolStripMenuItem->Size =
System::Drawing::Size(61, 20);
        this->довідкаToolStripMenuItem->Text = L"Довідка";
        //
        // проПрограмуToolStripMenuItem
        //
        this->проПрограмуToolStripMenuItem->Image =
(cli::safe_cast<System::Drawing::
            Image^>(resources-
        >GetObject(L"проПрограмуToolStripMenuItem.Image")));
        this->проПрограмуToolStripMenuItem->Name =
L"проПрограмуToolStripMenuItem";
        this->проПрограмуToolStripMenuItem->ShortcutKeys
            = System::Windows::Forms::Keys::F1;
        this->проПрограмуToolStripMenuItem->Size =
System::Drawing::Size(173, 22);
        this->проПрограмуToolStripMenuItem->Text = L"Про програму";
        this->проПрограмуToolStripMenuItem->Click += gcnew System::
            EventHandler(this,
        &MyForm::проПрограмуToolStripMenuItem_Click);
        //
        // toolStripПанельІнструментів
        //
        this->toolStripПанельІнструментів->Items->AddRange(gcnew cli
            ::array< System::Windows::Forms::ToolStripItem^  >(15)
```

```
            {
                this->toolStripButtonВідкрити,
                this->toolStripButtonЗберегти, this-
>toolStripSeparator1,
                this->toolStripButtonТест, this-
            >toolStripButtonРозрахувати,
                this->toolStripSeparator2,      this-
            >toolStripButtonГрафік,
                this->toolStripButtonЗвіт, this->toolStripButtonТаблиця,
                this->toolStripSeparator3, this-
            >toolStripButtonОчистити,
                this->toolStripSeparator4, this-
>toolStripButtonПроПрограму,
                this->toolStripSeparator5, this->toolStripButtonВихід
            });
            this->toolStripПанельІнструментів->Location =
System::Drawing::Point(0, 24);
            this->toolStripПанельІнструментів->Name =
L"toolStripПанельІнструментів";
            this->toolStripПанельІнструментів->Size =
System::Drawing::Size(639, 38);
            this->toolStripПанельІнструментів->TabIndex = 4;
            this->toolStripПанельІнструментів->Text = L"toolStrip1";
            //
            // toolStripButtonВідкрити
            //
            this->toolStripButtonВідкрити->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"toolStripButtonВідкрити.Image")));
            this->toolStripButtonВідкрити->ImageTransparentColor
                = System::Drawing::Color::Magenta;
            this->toolStripButtonВідкрити->Name =
L"toolStripButtonВідкрити";
            this->toolStripButtonВідкрити->Size =
System::Drawing::Size(68, 35);
            this->toolStripButtonВідкрити->Text = L"Відкрити...";
            this->toolStripButtonВідкрити->TextImageRelation
                =
            System::Windows::Forms::TextImageRelation::ImageAboveText;
            this->toolStripButtonВідкрити->ToolTipText = L"Відкрити файл";
            this->toolStripButtonВідкрити->Click += gcnew System
                ::EventHandler(this,
            &MyForm::відкритиФайлToolStripMenuItem_Click);
            //
            // toolStripButtonЗберегти
            //
            this->toolStripButtonЗберегти->Image =
(cli::safe_cast<System::Drawing
```

```
            ::Image^>(resources-
        >GetObject(L"toolStripButtonЗберегти.Image")));
        this->toolStripButtonЗберегти->ImageTransparentColor
            = System::Drawing::Color::Magenta;
        this->toolStripButtonЗберегти->Name =
L"toolStripButtonЗберегти";
        this->toolStripButtonЗберегти->Size =
System::Drawing::Size(70, 35);
        this->toolStripButtonЗберегти->Text = L"Зберегти...";
        this->toolStripButtonЗберегти->TextImageRelation
            =
        System::Windows::Forms::TextImageRelation::ImageAboveText;
        this->toolStripButtonЗберегти->Click += gcnew System
            ::EventHandler(this,
        &MyForm::зберегтиФайлToolStripMenuItem_Click);
        //
        // toolStripSeparator1
        //
        this->toolStripSeparator1->Name = L"toolStripSeparator1";
        this->toolStripSeparator1->Size = System::Drawing::Size(6,
38);
        //
        // toolStripButtonТест
        //
        this->toolStripButtonТест->Image =
(cli::safe_cast<System::Drawing
            ::Image^>(resources-
        >GetObject(L"toolStripButtonТест.Image")));
        this->toolStripButtonТест->ImageTransparentColor
            = System::Drawing::Color::Magenta;
        this->toolStripButtonТест->Name = L"toolStripButtonТест";
        this->toolStripButtonТест->Size = System::Drawing::Size(35,
35);
        this->toolStripButtonТест->Text = L"Тест";
        this->toolStripButtonТест->TextImageRelation
            =
        System::Windows::Forms::TextImageRelation::ImageAboveText;
        this->toolStripButtonТест->ToolTipText = L"Запуск тестового
розрахунку";
        this->toolStripButtonТест->Click += gcnew System
            ::EventHandler(this,
        &MyForm::тестовийПрикладToolStripMenuItem_Click);
        //
        // toolStripButtonРозрахувати
        //
        this->toolStripButtonРозрахувати->Image =
(cli::safe_cast<System::Drawing
            ::Image^>(resources-
        >GetObject(L"toolStripButtonРозрахувати.Image")));
```

```
            this->toolStripButtonРозрахувати->ImageTransparentColor
                = System::Drawing::Color::Magenta;
            this->toolStripButtonРозрахувати->Name =
L"toolStripButtonРозрахувати";
            this->toolStripButtonРозрахувати->Size =
System::Drawing::Size(74, 35);
            this->toolStripButtonРозрахувати->Text = L"Розрахунок";
            this->toolStripButtonРозрахувати->TextImageRelation
                =
            System::Windows::Forms::TextImageRelation::ImageAboveText;
            this->toolStripButtonРозрахувати->ToolTipText = L"Знайти
екстремум функції";
            this->toolStripButtonРозрахувати->Click += gcnew System
                ::EventHandler(this,
            &MyForm::розрахуватиToolStripMenuItem_Click);
            //
            // toolStripSeparator2
            //
            this->toolStripSeparator2->Name = L"toolStripSeparator2";
            this->toolStripSeparator2->Size = System::Drawing::Size(6,
38);
            //
            // toolStripButtonГрафік
            //
            this->toolStripButtonГрафік->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"toolStripButtonГрафік.Image")));
            this->toolStripButtonГрафік->ImageTransparentColor
                = System::Drawing::Color::Magenta;
            this->toolStripButtonГрафік->Name = L"toolStripButtonГрафік";
            this->toolStripButtonГрафік->Size = System::Drawing::Size(48,
35);
            this->toolStripButtonГрафік->Text = L"Графік";
            this->toolStripButtonГрафік->TextImageRelation
                =
            System::Windows::Forms::TextImageRelation::ImageAboveText;
            this->toolStripButtonГрафік->ToolTipText = L"Графік
залежності";
            this->toolStripButtonГрафік->Click += gcnew System
                ::EventHandler(this,
            &MyForm::графікToolStripMenuItem_Click);
            //
            // toolStripButtonЗвіт
            //
            this->toolStripButtonЗвіт->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"toolStripButtonЗвіт.Image")));
```

```
            this->toolStripButtonЗвіт->ImageTransparentColor
                 = System::Drawing::Color::Magenta;
            this->toolStripButtonЗвіт->Name = L"toolStripButtonЗвіт";
            this->toolStripButtonЗвіт->Size = System::Drawing::Size(32,
35);
            this->toolStripButtonЗвіт->Text = L"Звіт";
            this->toolStripButtonЗвіт->TextImageRelation
                  =
            System::Windows::Forms::TextImageRelation::ImageAboveText;
            this->toolStripButtonЗвіт->ToolTipText = L"Звіт за
результатами розрахунку";
            this->toolStripButtonЗвіт->Click += gcnew System
                 ::EventHandler(this,
            &MyForm::звітToolStripMenuItem_Click);
            //
            // toolStripButtonТаблиця
            //
            this->toolStripButtonТаблиця->Image =
(cli::safe_cast<System::Drawing
                 ::Image^>(resources-
            >GetObject(L"toolStripButtonТаблиця.Image")));
            this->toolStripButtonТаблиця->ImageTransparentColor
                 = System::Drawing::Color::Magenta;
            this->toolStripButtonТаблиця->Name =
L"toolStripButtonТаблиця";
            this->toolStripButtonТаблиця->Size = System::Drawing::Size(71,
35);
            this->toolStripButtonТаблиця->Text = L"Результати";
            this->toolStripButtonТаблиця->TextImageRelation
                 =
            System::Windows::Forms::TextImageRelation::ImageAboveText;
            this->toolStripButtonТаблиця->ToolTipText = L"Таблиця
результатів";
            this->toolStripButtonТаблиця->Click += gcnew System::
                 EventHandler(this,
            &MyForm::таблицяРозрахункуToolStripMenuItem_Click);
            //
            // toolStripSeparator3
            //
            this->toolStripSeparator3->Name = L"toolStripSeparator3";
            this->toolStripSeparator3->Size = System::Drawing::Size(6,
38);
            //
            // toolStripButtonОчистити
            //
            this->toolStripButtonОчистити->Image =
(cli::safe_cast<System::Drawing
                 ::Image^>(resources-
            >GetObject(L"toolStripButtonОчистити.Image")));
```

```
            this->toolStripButtonОчистити->ImageTransparentColor
                = System::Drawing::Color::Magenta;
            this->toolStripButtonОчистити->Name =
L"toolStripButtonОчистити";
            this->toolStripButtonОчистити->Size =
System::Drawing::Size(64, 35);
            this->toolStripButtonОчистити->Text = L"Очистити";
            this->toolStripButtonОчистити->TextImageRelation
                =
            System::Windows::Forms::TextImageRelation::ImageAboveText;
            this->toolStripButtonОчистити->ToolTipText = L"Очистити дані";
            this->toolStripButtonОчистити->Click += gcnew System::
                EventHandler(this,
            &MyForm::очиститиToolStripMenuItem_Click);
            //
            // toolStripSeparator4
            //
            this->toolStripSeparator4->Name = L"toolStripSeparator4";
            this->toolStripSeparator4->Size = System::Drawing::Size(6,
38);
            //
            // toolStripButtonПроПрограму
            //
            this->toolStripButtonПроПрограму->Image =
(cli::safe_cast<System::Drawing
                ::Image^>(resources-
            >GetObject(L"toolStripButtonПроПрограму.Image")));
            this->toolStripButtonПроПрограму->ImageTransparentColor
                = System::Drawing::Color::Magenta;
            this->toolStripButtonПроПрограму->Name =
L"toolStripButtonПроПрограму";
            this->toolStripButtonПроПрограму->Size =
System::Drawing::Size(91, 35);
            this->toolStripButtonПроПрограму->Text = L"Про програму";
            this->toolStripButtonПроПрограму->TextImageRelation
                =
            System::Windows::Forms::TextImageRelation::ImageAboveText;
            this->toolStripButtonПроПрограму->Click += gcnew System::
                EventHandler(this,
            &MyForm::проПрограмуToolStripMenuItem_Click);
            //
            // toolStripSeparator5
            //
            this->toolStripSeparator5->Name = L"toolStripSeparator5";
            this->toolStripSeparator5->Size = System::Drawing::Size(6,
38);
            //
            // toolStripButtonВихід
            //
```

```
            this->toolStripButtonВихід->Image =
(cli::safe_cast<System::Drawing
                  ::Image^>(resources-
            >GetObject(L"toolStripButtonВихід.Image")));
            this->toolStripButtonВихід->ImageTransparentColor
                  = System::Drawing::Color::Magenta;
            this->toolStripButtonВихід->Name = L"toolStripButtonВихід";
            this->toolStripButtonВихід->Size = System::Drawing::Size(39,
35);
            this->toolStripButtonВихід->Text = L"Вихід";
            this->toolStripButtonВихід->TextImageRelation
                  =
            System::Windows::Forms::TextImageRelation::ImageAboveText;
            this->toolStripButtonВихід->Click += gcnew System::
                  EventHandler(this,
            &MyForm::вихідToolStripMenuItem_Click);
            //
            // panelГоловна
            //
            this->panelГоловна->Controls->Add(this->tabControlСторінки);
            this->panelГоловна->Dock =
System::Windows::Forms::DockStyle::Fill;
            this->panelГоловна->Location = System::Drawing::Point(0, 62);
            this->panelГоловна->Name = L"panelГоловна";
            this->panelГоловна->Size = System::Drawing::Size(639, 262);
            this->panelГоловна->TabIndex = 5;
            //
            // tabControlСторінки
            //
            this->tabControlСторінки->Controls->Add(this->tabPageДані);
            this->tabControlСторінки->Controls->Add(this-
>tabPageРезультат);
            this->tabControlСторінки->Controls->Add(this->tabPageГрафік);
            this->tabControlСторінки->Controls->Add(this-
>tabPageІтерації);
            this->tabControlСторінки->Dock =
System::Windows::Forms::DockStyle::Fill;
            this->tabControlСторінки->Location = System::Drawing::Point(0,
0);
            this->tabControlСторінки->Name = L"tabControlСторінки";
            this->tabControlСторінки->SelectedIndex = 0;
            this->tabControlСторінки->Size = System::Drawing::Size(639,
262);
            this->tabControlСторінки->TabIndex = 1;
            this->tabControlСторінки->SelectedIndexChanged += gcnew System
                  ::EventHandler(this,
            &MyForm::tabControlСторінки_SelectedIndexChanged);
            //
            // tabPageДані
```

```
            //
            this->tabPageДані->ContextMenuStrip = this-
>contextMenuStripВихідніДані;
            this->tabPageДані->Controls->Add(this-
>panelКоефіцієнтиТочність);
            this->tabPageДані->Controls->Add(this-
>panelЕкстремумІнтервал);
            this->tabPageДані->Location = System::Drawing::Point(4, 22);
            this->tabPageДані->Name = L"tabPageДані";
            this->tabPageДані->Padding =
System::Windows::Forms::Padding(3);
            this->tabPageДані->Size = System::Drawing::Size(631, 236);
            this->tabPageДані->TabIndex = 0;
            this->tabPageДані->Text = L"Вихідні дані";
            this->tabPageДані->UseVisualStyleBackColor = true;
            //
            // panelКоефіцієнтиТочність
            //
            this->panelКоефіцієнтиТочність->Controls->Add(this-
>groupBoxКоефіцієнти);
            this->panelКоефіцієнтиТочність->Controls->Add(this-
>groupBoxТочність);
            this->panelКоефіцієнтиТочність->Dock
                  = System::Windows::Forms::DockStyle::Fill;
            this->panelКоефіцієнтиТочність->Location =
System::Drawing::Point(253, 3);
            this->panelКоефіцієнтиТочність->Name =
L"panelКоефіцієнтиТочність";
            this->panelКоефіцієнтиТочність->Size =
System::Drawing::Size(375, 230);
            this->panelКоефіцієнтиТочність->TabIndex = 14;
            //
            // groupBoxКоефіцієнти
            //
            this->groupBoxКоефіцієнти->Controls->Add(this->textBox_k5);
            this->groupBoxКоефіцієнти->Controls->Add(this->textBox_k4);
            this->groupBoxКоефіцієнти->Controls->Add(this->label_k4);
            this->groupBoxКоефіцієнти->Controls->Add(this->textBox_k3);
            this->groupBoxКоефіцієнти->Controls->Add(this->textBox_k2);
            this->groupBoxКоефіцієнти->Controls->Add(this->label_k2);
            this->groupBoxКоефіцієнти->Controls->Add(this->textBox_k1);
            this->groupBoxКоефіцієнти->Controls->Add(this->label_k1);
            this->groupBoxКоефіцієнти->Controls->Add(this->label_k3);
            this->groupBoxКоефіцієнти->Dock =
System::Windows::Forms::DockStyle::Fill;
            this->groupBoxКоефіцієнти->Location =
System::Drawing::Point(0, 0);
            this->groupBoxКоефіцієнти->Name = L"groupBoxКоефіцієнти";
```

```
            this->groupBoxКоефіцієнти->Size = System::Drawing::Size(375,
130);
            this->groupBoxКоефіцієнти->TabIndex = 14;
            this->groupBoxКоефіцієнти->TabStop = false;
            this->groupBoxКоефіцієнти->Text = L"Коефіцієнти рівняння";
            //
            // textBox_k5
            //
            this->textBox_k5->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif",
                10, System::Drawing::FontStyle::Bold,
                System::Drawing::GraphicsUnit::Point,
                static_cast<System::Byte>(204)));
            this->textBox_k5->ForeColor =
System::Drawing::SystemColors::GrayText;
            this->textBox_k5->Location = System::Drawing::Point(296, 49);
            this->textBox_k5->Name = L"textBox_k5";
            this->textBox_k5->Size = System::Drawing::Size(38, 23);
            this->textBox_k5->TabIndex = 14;
            this->textBox_k5->Text = L"k5";
            this->textBox_k5->TextAlign
                = System::Windows::Forms::HorizontalAlignment::Center;
            //
            // textBox_k4
            //
            this->textBox_k4->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif",
                10, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit
                ::Point, static_cast<System::Byte>(204)));
            this->textBox_k4->ForeColor =
System::Drawing::SystemColors::GrayText;
            this->textBox_k4->Location = System::Drawing::Point(226, 58);
            this->textBox_k4->Name = L"textBox_k4";
            this->textBox_k4->Size = System::Drawing::Size(38, 23);
            this->textBox_k4->TabIndex = 13;
            this->textBox_k4->Text = L"k4";
            this->textBox_k4->TextAlign
                = System::Windows::Forms::HorizontalAlignment::Center;
            //
            // label_k4
            //
            this->label_k4->AutoSize = true;
            this->label_k4->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif",
                10, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit
                ::Point, static_cast<System::Byte>(204)));
            this->label_k4->Location = System::Drawing::Point(270, 61);
```

```
            this->label_k4->Name = L"label_k4";
            this->label_k4->Size = System::Drawing::Size(29, 17);
            this->label_k4->TabIndex = 12;
            this->label_k4->Text = L"* X";
            //
            // textBox_k3
            //
            this->textBox_k3->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif",
                10, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit
                ::Point, static_cast<System::Byte>(204)));
            this->textBox_k3->ForeColor =
System::Drawing::SystemColors::HotTrack;
            this->textBox_k3->Location = System::Drawing::Point(166, 49);
            this->textBox_k3->Name = L"textBox_k3";
            this->textBox_k3->Size = System::Drawing::Size(38, 23);
            this->textBox_k3->TabIndex = 11;
            this->textBox_k3->Text = L"k3";
            this->textBox_k3->TextAlign
                = System::Windows::Forms::HorizontalAlignment::Center;
            //
            // textBox_k2
            //
            this->textBox_k2->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif",
                10, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit
                ::Point, static_cast<System::Byte>(204)));
            this->textBox_k2->ForeColor =
System::Drawing::SystemColors::HotTrack;
            this->textBox_k2->Location = System::Drawing::Point(98, 58);
            this->textBox_k2->Name = L"textBox_k2";
            this->textBox_k2->Size = System::Drawing::Size(38, 23);
            this->textBox_k2->TabIndex = 9;
            this->textBox_k2->Text = L"k2";
            this->textBox_k2->TextAlign
                = System::Windows::Forms::HorizontalAlignment::Center;
            //
            // label_k2
            //
            this->label_k2->AutoSize = true;
            this->label_k2->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif",
                10, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit
                ::Point, static_cast<System::Byte>(204)));
            this->label_k2->Location = System::Drawing::Point(142, 61);
            this->label_k2->Name = L"label_k2";
```

```
            this->label_k2->Size = System::Drawing::Size(29, 17);
            this->label_k2->TabIndex = 8;
            this->label_k2->Text = L"* X";
            //
            // textBox_k1
            //
            this->textBox_k1->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif",
                10, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit
                ::Point, static_cast<System::Byte>(204)));
            this->textBox_k1->ForeColor =
System::Drawing::SystemColors::HotTrack;
            this->textBox_k1->Location = System::Drawing::Point(41, 58);
            this->textBox_k1->Name = L"textBox_k1";
            this->textBox_k1->Size = System::Drawing::Size(38, 23);
            this->textBox_k1->TabIndex = 7;
            this->textBox_k1->Text = L"k1";
            this->textBox_k1->TextAlign
                = System::Windows::Forms::HorizontalAlignment::Center;
            //
            // label_k1
            //
            this->label_k1->AutoSize = true;
            this->label_k1->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif",
                10, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit
                ::Point, static_cast<System::Byte>(204)));
            this->label_k1->Location = System::Drawing::Point(80, 61);
            this->label_k1->Name = L"label_k1";
            this->label_k1->Size = System::Drawing::Size(22, 17);
            this->label_k1->TabIndex = 6;
            this->label_k1->Text = L"+ ";
            //
            // label_k3
            //
            this->label_k3->AutoSize = true;
            this->label_k3->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif",
                10, System::Drawing::FontStyle::Bold,
System::Drawing::GraphicsUnit
                ::Point, static_cast<System::Byte>(204)));
            this->label_k3->Location = System::Drawing::Point(210, 58);
            this->label_k3->Name = L"label_k3";
            this->label_k3->Size = System::Drawing::Size(22, 17);
            this->label_k3->TabIndex = 15;
            this->label_k3->Text = L"+ ";
            //
```

```
            // groupBoxТочність
            //
            this->groupBoxТочність->Controls->Add(this->textBox_e);
            this->groupBoxТочність->Controls->Add(this->label_e);
            this->groupBoxТочність->Dock =
System::Windows::Forms::DockStyle::Bottom;
            this->groupBoxТочність->Location = System::Drawing::Point(0,
130);
            this->groupBoxТочність->Name = L"groupBoxТочність";
            this->groupBoxТочність->Size = System::Drawing::Size(375,
100);
            this->groupBoxТочність->TabIndex = 1;
            this->groupBoxТочність->TabStop = false;
            this->groupBoxТочність->Text = L"Точність пошуку екстремума";
            //
            // textBox_e
            //
            this->textBox_e->Location = System::Drawing::Point(9, 44);
            this->textBox_e->Name = L"textBox_e";
            this->textBox_e->Size = System::Drawing::Size(105, 20);
            this->textBox_e->TabIndex = 13;
            //
            // label_e
            //
            this->label_e->AutoSize = true;
            this->label_e->Location = System::Drawing::Point(9, 28);
            this->label_e->Name = L"label_e";
            this->label_e->Size = System::Drawing::Size(206, 13);
            this->label_e->TabIndex = 12;
            this->label_e->Text = L"Задайте точність пошуку екстремума e";
            //
            // panelЕкстремумІнтервал
            //
            this->panelЕкстремумІнтервал->Controls->Add(this-
>groupBoxІнтервалПошуку);
            this->panelЕкстремумІнтервал->Controls->Add(this-
>groupBoxЕкстремум);
            this->panelЕкстремумІнтервал->Dock =
System::Windows::Forms::DockStyle::Left;
            this->panelЕкстремумІнтервал->Location =
System::Drawing::Point(3, 3);
            this->panelЕкстремумІнтервал->MinimumSize =
System::Drawing::Size(200, 0);
            this->panelЕкстремумІнтервал->Name =
L"panelЕкстремумІнтервал";
            this->panelЕкстремумІнтервал->Size =
System::Drawing::Size(250, 230);
            this->panelЕкстремумІнтервал->TabIndex = 13;
            //
```

```
            // groupBoxІнтервалПошуку
            //
            this->groupBoxІнтервалПошуку->Controls->Add(this->textBox_b);
            this->groupBoxІнтервалПошуку->Controls->Add(this->label_b);
            this->groupBoxІнтервалПошуку->Controls->Add(this->textBox_a);
            this->groupBoxІнтервалПошуку->Controls->Add(this->label_a);
            this->groupBoxІнтервалПошуку->Dock =
System::Windows::Forms::DockStyle::Fill;
            this->groupBoxІнтервалПошуку->Location =
System::Drawing::Point(0, 81);
            this->groupBoxІнтервалПошуку->Name =
L"groupBoxІнтервалПошуку";
            this->groupBoxІнтервалПошуку->Size =
System::Drawing::Size(250, 149);
            this->groupBoxІнтервалПошуку->TabIndex = 15;
            this->groupBoxІнтервалПошуку->TabStop = false;
            this->groupBoxІнтервалПошуку->Text = L"Інтервал пошуку
екстремума";
            //
            // textBox_b
            //
            this->textBox_b->Location = System::Drawing::Point(10, 80);
            this->textBox_b->Name = L"textBox_b";
            this->textBox_b->Size = System::Drawing::Size(105, 20);
            this->textBox_b->TabIndex = 13;
            //
            // label_b
            //
            this->label_b->AutoSize = true;
            this->label_b->Location = System::Drawing::Point(6, 65);
            this->label_b->Name = L"label_b";
            this->label_b->Size = System::Drawing::Size(171, 13);
            this->label_b->TabIndex = 12;
            this->label_b->Text = L"Задайте праву межу інтервалу b";
            //
            // textBox_a
            //
            this->textBox_a->Location = System::Drawing::Point(10, 41);
            this->textBox_a->Name = L"textBox_a";
            this->textBox_a->Size = System::Drawing::Size(105, 20);
            this->textBox_a->TabIndex = 11;
            //
            // label_a
            //
            this->label_a->AutoSize = true;
            this->label_a->Location = System::Drawing::Point(6, 26);
            this->label_a->Name = L"label_a";
            this->label_a->Size = System::Drawing::Size(161, 13);
            this->label_a->TabIndex = 10;
```

```
            this->label_a->Text = L"Задайте ліву межу інтервалу a";
            //
            // groupBoxЕкстремум
            //
            this->groupBoxЕкстремум->Controls->Add(this-
>checkedListBoxЕкстремум);
            this->groupBoxЕкстремум->Dock =
System::Windows::Forms::DockStyle::Top;
            this->groupBoxЕкстремум->Location = System::Drawing::Point(0,
0);
            this->groupBoxЕкстремум->Name = L"groupBoxЕкстремум";
            this->groupBoxЕкстремум->Size = System::Drawing::Size(250,
81);
            this->groupBoxЕкстремум->TabIndex = 14;
            this->groupBoxЕкстремум->TabStop = false;
            this->groupBoxЕкстремум->Text = L"Знаходження екстремуму
функції";
            //
            // checkedListBoxЕкстремум
            //
            this->checkedListBoxЕкстремум->Cursor =
System::Windows::Forms::Cursors::Hand;
            this->checkedListBoxЕкстремум->Dock =
System::Windows::Forms::DockStyle::Fill;
            this->checkedListBoxЕкстремум->FormattingEnabled = true;
            this->checkedListBoxЕкстремум->Items->AddRange(gcnew
cli::array< System
                  ::Object^  >(2) { L"Максимум функції", L"Мінімум
            функції" });
            this->checkedListBoxЕкстремум->Location =
System::Drawing::Point(3, 16);
            this->checkedListBoxЕкстремум->Name =
L"checkedListBoxЕкстремум";
            this->checkedListBoxЕкстремум->Size =
System::Drawing::Size(244, 62);
            this->checkedListBoxЕкстремум->TabIndex = 6;
            //
            // tabPageРезультат
            //
            this->tabPageРезультат->ContextMenuStrip = this-
>contextMenuStripРезультати;
            this->tabPageРезультат->Controls->Add(this->richTextBoxЗвіт);
            this->tabPageРезультат->Location = System::Drawing::Point(4,
22);
            this->tabPageРезультат->Name = L"tabPageРезультат";
            this->tabPageРезультат->Padding =
System::Windows::Forms::Padding(3);
            this->tabPageРезультат->Size = System::Drawing::Size(631,
236);
```

```
            this->tabPageРезультат->TabIndex = 1;
            this->tabPageРезультат->Text = L"Результати розрахунку";
            this->tabPageРезультат->UseVisualStyleBackColor = true;
            //
            // richTextBoxЗвіт
            //
            this->richTextBoxЗвіт->ContextMenuStrip = this-
>contextMenuStripРезультати;
            this->richTextBoxЗвіт->Dock =
System::Windows::Forms::DockStyle::Fill;
            this->richTextBoxЗвіт->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans
                Serif", 9, System::Drawing::FontStyle::Regular,
System::Drawing::
                GraphicsUnit::Point, static_cast<System::Byte>(204)));
            this->richTextBoxЗвіт->Location = System::Drawing::Point(3,
3);
            this->richTextBoxЗвіт->Name = L"richTextBoxЗвіт";
            this->richTextBoxЗвіт->Size = System::Drawing::Size(625, 230);
            this->richTextBoxЗвіт->TabIndex = 0;
            this->richTextBoxЗвіт->Text = L"";
            //
            // tabPageГрафік
            //
            this->tabPageГрафік->ContextMenuStrip = this-
>contextMenuStripГрафік;
            this->tabPageГрафік->Controls->Add(this->chartГрафік);
            this->tabPageГрафік->Controls->Add(this->pictureBox1);
            this->tabPageГрафік->Location = System::Drawing::Point(4, 22);
            this->tabPageГрафік->Name = L"tabPageГрафік";
            this->tabPageГрафік->Padding =
System::Windows::Forms::Padding(3);
            this->tabPageГрафік->Size = System::Drawing::Size(631, 236);
            this->tabPageГрафік->TabIndex = 2;
            this->tabPageГрафік->Text = L"Графічна залежність";
            this->tabPageГрафік->UseVisualStyleBackColor = true;
            //
            // chartГрафік
            //
            chartArea1->Name = L"ChartArea1";
            this->chartГрафік->ChartAreas->Add(chartArea1);
            this->chartГрафік->Dock =
System::Windows::Forms::DockStyle::Fill;
            legend1->Name = L"Legend1";
            this->chartГрафік->Legends->Add(legend1);
            this->chartГрафік->Location = System::Drawing::Point(3, 3);
            this->chartГрафік->Name = L"chartГрафік";
            series1->ChartArea = L"ChartArea1";
            series1->Legend = L"Legend1";
```

```
            series1->Name = L"Series1";
            this->chartГрафік->Series->Add(series1);
            this->chartГрафік->Size = System::Drawing::Size(625, 230);
            this->chartГрафік->TabIndex = 3;
            this->chartГрафік->Text = L"chart1";
            this->chartГрафік->Visible = false;
            //
            // pictureBox1
            //
            this->pictureBox1->Dock =
System::Windows::Forms::DockStyle::Fill;
            this->pictureBox1->Location = System::Drawing::Point(3, 3);
            this->pictureBox1->Name = L"pictureBox1";
            this->pictureBox1->Size = System::Drawing::Size(625, 230);
            this->pictureBox1->SizeMode
                 = System::Windows::Forms::PictureBoxSizeMode::Zoom;
            this->pictureBox1->TabIndex = 2;
            this->pictureBox1->TabStop = false;
            //
            // tabPageІтерації
            //
            this->tabPageІтерації->Controls->Add(this-
>dataGridViewІтерації);
            this->tabPageІтерації->Location = System::Drawing::Point(4,
22);
            this->tabPageІтерації->Name = L"tabPageІтерації";
            this->tabPageІтерації->Padding =
System::Windows::Forms::Padding(3);
            this->tabPageІтерації->Size = System::Drawing::Size(631, 236);
            this->tabPageІтерації->TabIndex = 3;
            this->tabPageІтерації->Text = L"Хід розрахунку";
            this->tabPageІтерації->UseVisualStyleBackColor = true;
            //
            // dataGridViewІтерації
            //
            this->dataGridViewІтерації->ColumnHeadersHeightSizeMode =
System::
                 Windows::Forms::DataGridViewColumnHeadersHeightSizeMode:
            :AutoSize;
            this->dataGridViewІтерації->Dock =
System::Windows::Forms::DockStyle::Fill;
            this->dataGridViewІтерації->Location =
System::Drawing::Point(3, 3);
            this->dataGridViewІтерації->Name = L"dataGridViewІтерації";
            this->dataGridViewІтерації->Size = System::Drawing::Size(625,
230);
            this->dataGridViewІтерації->TabIndex = 0;
            //
            // MyForm
```

```
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
            this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
            this->AutoScroll = true;
            this->ClientSize = System::Drawing::Size(639, 324);
            this->Controls->Add(this->panelГоловна);
            this->Controls->Add(this->toolStripПанельІнструментів);
            this->Controls->Add(this->menuStripГоловнеМеню);
            this->MainMenuStrip = this->menuStripГоловнеМеню;
            this->MinimumSize = System::Drawing::Size(655, 350);
            this->Name = L"MyForm";
            this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScreen;
            this->Text = L"Пошук екстремума функції методом золотого
перетину";
            this->FormClosing += gcnew System::Windows::Forms
                  ::FormClosingEventHandler(this,
            &MyForm::MyForm_FormClosing);
            this->Load += gcnew System::EventHandler(this,
&MyForm::MyForm_Load);
            this->Resize += gcnew System::EventHandler(this,
&MyForm::MyForm_Resize);
            this->contextMenuStripВихідніДані->ResumeLayout(false);
            this->contextMenuStripРезультати->ResumeLayout(false);
            this->contextMenuStripГрафік->ResumeLayout(false);
            this->menuStripГоловнеМеню->ResumeLayout(false);
            this->menuStripГоловнеМеню->PerformLayout();
            this->toolStripПанельІнструментів->ResumeLayout(false);
            this->toolStripПанельІнструментів->PerformLayout();
            this->panelГоловна->ResumeLayout(false);
            this->tabControlСторінки->ResumeLayout(false);
            this->tabPageДані->ResumeLayout(false);
            this->panelКоефіцієнтиТочність->ResumeLayout(false);
            this->groupBoxКоефіцієнти->ResumeLayout(false);
            this->groupBoxКоефіцієнти->PerformLayout();
            this->groupBoxТочність->ResumeLayout(false);
            this->groupBoxТочність->PerformLayout();
            this->panelЕкстремумІнтервал->ResumeLayout(false);
            this->groupBoxІнтервалПошуку->ResumeLayout(false);
            this->groupBoxІнтервалПошуку->PerformLayout();
            this->groupBoxЕкстремум->ResumeLayout(false);
            this->tabPageРезультат->ResumeLayout(false);
            this->tabPageГрафік->ResumeLayout(false);

      (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
                  ->chartГрафік))->EndInit();

      (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
```

```
                ->pictureBox1))->EndInit();
            this->tabPageІтерації->ResumeLayout(false);

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this
                ->dataGridViewІтерації))->EndInit();
            this->ResumeLayout(false);
            this->PerformLayout();

        }


#pragma endregion


private: System::Void розрахуватиToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
    float k1, k2, k3, k4, k5, a, b, eps;
    bool mmaxx;
    try
    {
        if (!String::IsNullOrEmpty(textBox_a->Text)) a =
            float::Parse(textBox_a->Text);
        if (!String::IsNullOrEmpty(textBox_b->Text)) b =
            float::Parse(textBox_b->Text);
        if (!String::IsNullOrEmpty(textBox_e->Text)) eps =
            float::Parse(textBox_e->Text);
        if (checkedListBoxЕкстремум->GetItemChecked(1))
        {
            mmaxx = true;
        }
        else
        {
            mmaxx = false;
            checkedListBoxЕкстремум->SetItemChecked(0, true);
        }
        if ((textBox_k1->Text != "k1") && (textBox_k5->Text != "k5"))
        {
            if (!String::IsNullOrEmpty(textBox_k1->Text)) k1 =
                float::Parse(textBox_k1->Text);
            if (!String::IsNullOrEmpty(textBox_k2->Text)) k2 =
                float::Parse(textBox_k2->Text);
            if (!String::IsNullOrEmpty(textBox_k3->Text)) k3 =
                float::Parse(textBox_k3->Text);
            if (!String::IsNullOrEmpty(textBox_k4->Text)) k4 =
                float::Parse(textBox_k4->Text);
            if (!String::IsNullOrEmpty(textBox_k5->Text)) k5 =
                float::Parse(textBox_k5->Text);
            Екстремум = gcnew MyClass(k1, k2, k3, k4, k5, a, b, eps,
mmaxx);
```

```
            }
            else if ((textBox_k1->Text != "k1") && (textBox_k5->Text ==
"k5"))
            {
                  if (!String::IsNullOrEmpty(textBox_k1->Text)) k1 =
                        float::Parse(textBox_k1->Text);
                  if (!String::IsNullOrEmpty(textBox_k2->Text)) k2 =
                        float::Parse(textBox_k2->Text);
                  if (!String::IsNullOrEmpty(textBox_k3->Text)) k3 =
                        float::Parse(textBox_k3->Text);
                  Екстремум = gcnew MyClass(k1, k2, k3, a, b, mmaxx);
            }
            // Виклик функції розрахунку Method() та виведення
повідомлення
            //з результатом, а також запитом на виведення прміжних
      результатів:
            if ((MessageBox::Show(Екстремум->Method() + "\n Вивести
таблицю
                  проміжних розрахунків?", "Екстремум функції",
                  MessageBoxButtons::YesNo, MessageBoxIcon::Question))
                  == ::DialogResult::Yes)
                  таблицяРозрахункуToolStripMenuItem_Click(this, e);
      }
      catch (Exception^ Ситуація)
      {
            MessageBox::Show("Перевірте введені дані!\n" + Ситуація-
>Message,
                  "Помилкове введення", MessageBoxButtons::OK,
                  MessageBoxIcon::Exclamation);
      }
}

private: System::Void очиститиToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
      if (tabControlСторінки->SelectedIndex == 0) //Перевірка чи активна
перша вкладка
      {
            // Очищення полів введення даних:
            textBox_k1->Text = "k1";
            textBox_k2->Text = "k2";
            textBox_k3->Text = "k3";
            textBox_k4->Text = "k4";
            textBox_k5->Text = "k5";
            textBox_a->Text = nullptr;
            textBox_b->Text = nullptr;
            textBox_e->Text = nullptr;
            checkedListBoxЕкстремум->SelectedIndex = -1;
            checkedListBoxЕкстремум->SetItemChecked(0, false);
            checkedListBoxЕкстремум->SetItemChecked(1, false);
```

```
        }
        else if (tabControlСторінки->SelectedIndex == 1)
        {
            видалитиToolStripMenuItem_Click(this, e);
        }

    }

    private: System::Void
    тестовийПрикладToolStripMenuItem_Click(System::Object^  sender,
    System::EventArgs^  e) {
        Екстремум = gcnew MyClass();
        textBox_k1->Text = "15";
        textBox_k2->Text = "5";
        textBox_k3->Text = "2";
        textBox_k4->Text = "0";
        textBox_k5->Text = "0";
        textBox_a->Text = "-100";
        textBox_b->Text = "100";
        textBox_e->Text = "0.001";
        checkedListBoxЕкстремум->SetItemChecked(1, true);
        // Виклик функції розрахунку Method() та виведення повідомлення з
    результатом,
        // а також запитом на виведення прміжних результатів:
        if ((MessageBox::Show(Екстремум->Method() + "\n Вивести таблицю
    проміжних
            розрахунків?", "Екстремум функції", MessageBoxButtons::YesNo,
            MessageBoxIcon::Question)) == ::DialogResult::Yes)
            таблицяРозрахункуToolStripMenuItem_Click(this, e);
    }

    private: System::Void відкритиФайлToolStripMenuItem_Click(System::Object^
    sender, System::EventArgs^  e) {
        //Перевірка чи активна третя вкладка:
        if (tabControlСторінки->SelectedIndex == 2)
        {//Тоді відкриваємо графічний файл:
            //Створюємо об'єкт класу OpenFileDialog:
            OpenFileDialog ^ openFileDialog1 = gcnew OpenFileDialog();
            //Встановлюємо в діалоговому вікні фільтр для відкриття
    файлів:
            openFileDialog1->Filter = L"Файли зображень |
        *.bmp;*.jpg;*.gif; *.png";
            //Якщо користувач обрав в діалоговому вікні файл і його ім'я
    не пусте:
            if (openFileDialog1->ShowDialog() ==
        System::Windows::Forms::DialogResult::OK
                && openFileDialog1->FileName->Length > 0)
            {
                //Відкриваємо графічний файл в об'єкті pictureBox1:
```

```
                pictureBox1->Image = Image::FromFile(openFileDialog1-
            >FileName);
                //Розміщуємо ім'я відкритого файлу в заголовку вкладки:
                tabPageГрафік->Text = Path::GetFileName(openFileDialog1-
            >FileName);
                chartГрафік->Visible = false;
            }
        }
        else if (tabControlСторінки->SelectedIndex == 1)//Активна друга
вкладка
        {//Тоді відкриваємо файл *.rtf
            //Створюємо об'єкт класу OpenFileDialog:
            OpenFileDialog ^ openFileDialog1 = gcnew OpenFileDialog();
            //Встановлюємо в діалоговому вікні фільтр для відкриття
файлів:
            openFileDialog1->Filter = L"Текстові файли | *.rtf";
            //Якщо користувач обрав в діалоговому вікні файл і його ім'я
не пусте:
            if (openFileDialog1->ShowDialog() ==
        System::Windows::Forms::DialogResult::OK
                && openFileDialog1->FileName->Length > 0)
            {
                //Відкриваємо файл формату RTF в об'єкті richTextBox1:
                richTextBoxЗвіт->LoadFile(openFileDialog1->FileName);
                //Розміщуємо ім'я відкритого файлу в заголовку вкладки:
                tabPageРезультат->Text =
            Path::GetFileName(openFileDialog1->FileName);
            }
        }
}

private: System::Void зберегтиФайлToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
        if (tabControlСторінки->SelectedIndex == 2) //Перевірка чи активна
перша вкладка
        {//Тоді зберігаємо графічний файл
        // Відобразити вікно SaveFileDialog щоб користувач міг зберегти
зображення
            SaveFileDialog ^ saveFileDialog1 = gcnew SaveFileDialog();
            saveFileDialog1->Filter =
                "Зображення JPeg|*.jpg|Зображення
Bitmap|*.bmp|Зображення Gif|*.gif";
            saveFileDialog1->Title = "Зберегти файл зображення";
            saveFileDialog1->ShowDialog();
            // Якщо ім'я файлу не пустий рядок, зберегти зображення:
            if (saveFileDialog1->FileName != "")
            {
        // Зберегти зображення через System::IO::FileStream, створений
методом OpenFile:
```

```
                ::IO::FileStream ^ fs =
                        safe_cast<System::IO::FileStream^>(saveFileDialog1-
>OpenFile());
        // Запам'ятовування графічного формату файлу
System::Drawing::Imaging::ImageFormat
                ::Imaging::ImageFormat^ ГрафічнийФормат;
                // Можна зберегти зображення у відповідному форматі
ImageFormat на
                // основі типу файлу, обраного в діалоговому вікні.
                // Тип файлу береться з властивості FilterIndex
                switch (saveFileDialog1->FilterIndex)
                {
                case 1:
                        ГрафічнийФормат = ::Imaging::ImageFormat::Jpeg;
                        break;
                case 2:
                        ГрафічнийФормат = ::Imaging::ImageFormat::Bmp;
                        break;
                case 3:
                        ГрафічнийФормат = ::Imaging::ImageFormat::Gif;
                        break;
                }
                // Перевіряємо чи створена діаграма (графік):
                if (chartГрафік->Visible == true)
                {
                        // Зберігаємо діаграму:
                        chartГрафік->SaveImage(fs, ГрафічнийФормат);
                }
                // Інакше зберігаємо відкритий малюнок:
                else pictureBox1->Image->Save(fs, ГрафічнийФормат);
                //Розміщуємо ім'я файлу в заголовку вкладки:
                tabControlСторінки->SelectedTab->Text =
                        Path::GetFileName(saveFileDialog1->FileName);
                fs->Close();
        }
    }
    else if (tabControlСторінки->SelectedIndex == 1)//Активна друга
вкладка
    {//Тоді зберігаємо файл *.rtf
        //Створюємо об'єкт класу SaveFileDialog:
        SaveFileDialog ^ saveFileDialog1 = gcnew SaveFileDialog();
        //Встановлюємо в діалоговому вікні фільтр для збереження
файлів:
        saveFileDialog1->Filter = L"Текстові файли | *.rtf";
        //Якщо користувач обрав в діалоговому вікні файл і його ім'я
не пусте:
        if (saveFileDialog1->ShowDialog() ==
    System::Windows::Forms::DialogResult::OK
                && saveFileDialog1->FileName->Length > 0)
```

```
            {
                //Зберігаємо файл формату RTF з об'єкту richTextBox1:
                richTextBoxЗвіт->SaveFile(saveFileDialog1->FileName);
                //Розміщуємо ім'я файлу в заголовку вкладки:
                tabPageРезультат->Text =
            Path::GetFileName(saveFileDialog1->FileName);
            }
        }
}

private: System::Void MyForm_Resize(System::Object^  sender,
System::EventArgs^  e) {
        //Вирівнювання ширини панелей в межах форми:
        panelЕкстремумІнтервал->Width = panelКоефіцієнтиТочність->Width -
170;
}

private: System::Void вихідToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
        Close();//Закриття форми
}

private: System::Void виділитиВсеToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
        // Виділяє весь текст:
        richTextBoxЗвіт->SelectAll();
}

private: System::Void скопіюватиToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
        if (richTextBoxЗвіт->SelectionLength > 0)
        {
            // Копіює виділений текст в буфер обміну:
            richTextBoxЗвіт->Copy();
        }
}

private: System::Void вирізатиToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
        // Перевіряємо чи є виділений текст
        if (!richTextBoxЗвіт->SelectedText->Equals(""))
        {
            //Вирізає виділений текст та вставляє його в буфер обміну:
            richTextBoxЗвіт->Cut();
        }
}

private: System::Void видалитиToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
```

```
        //Очищує текст (видаляє) в об'єкті richTextBox1:
        richTextBoxЗвіт->Clear();
}

private: System::Void вставитиToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
        //Вставляє текст з буферу обміну:
        richTextBoxЗвіт->Paste();
}

private: System::Void відмінитиToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
        if (richTextBoxЗвіт->CanUndo == true)
        {
                //Відміняє останню операцію:
                richTextBoxЗвіт->Undo();
                //Очищує буфер undo від збереженої попередньої операції:
                richTextBoxЗвіт->ClearUndo();
        }
}

private: System::Void проПрограмуToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
        MessageBox::Show("Програма знаходження екстремума функції методом
золотого перетину",
                "Про програму", MessageBoxButtons::OK,
        MessageBoxIcon::Information);
}

private: System::Void графікToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
        chartГрафік->DataSource = Екстремум->ТаблицяЗначень();
        //Прив'язка графіка до джерела даних:
        chartГрафік->DataBind();
        //На горизонтальній осі відкладаємо значення x:
        chartГрафік->Series["Series1"]->XValueMember = "Значення x";
        //А по вертикальній осі відкладаємо значення z:
        chartГрафік->Series["Series1"]->YValueMembers = "Значення функції
y(x)";
        //Задаємо тип діаграми - лінійна:
        chartГрафік->Series["Series1"]->ChartType = SeriesChartType::Line;
        //Тип діаграми може бути іншим, наприклад: Pie, Column і ін.
        chartГрафік->Series["Series1"]->Color = Color::Aqua;
        //Легенду на графіку не відображаємо:
        chartГрафік->Series["Series1"]->IsVisibleInLegend = false;
        chartГрафік->Visible = true;
        tabControlСторінки->SelectedIndex = 2;
}
```

```
private: System::Void звітToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
      //Виведення результатів розрахунку в текстове поле richTextBoxЗвіт:
      richTextBoxЗвіт->AppendText(Екстремум->ТекстовийЗвіт());
      tabControlСторінки->SelectedIndex = 1;
      tabPageРезультат->Text = "Результати розрахунку";
}

private: System::Void
таблицяРозрахункуToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e) {
      //Виведення результатів розрахунку в таблицю dataGridViewІтерації:
      dataGridViewІтерації->DataSource = Екстремум->ТаблицяРозрахунків();
      tabControlСторінки->SelectedIndex = 3;
}

private: System::Void MyForm_FormClosing(System::Object^  sender,
System::Windows::Forms::FormClosingEventArgs^  e) {
      //Якщо натиснуте No, не закривати форму:
      if ((MessageBox::Show("Вийти з програми?", "Вихід",
MessageBoxButtons::YesNo,
            MessageBoxIcon::Question)) == ::DialogResult::No) e->Cancel =
true;
}

private: System::Void
tabControlСторінки_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
      if (tabControlСторінки->SelectedIndex == 0) //Перевірка чи активна
перша вкладка
      {
            toolStripButtonЗберегти->Enabled = false;
            зберегтиФайлToolStripMenuItem->Enabled = false;
            toolStripButtonТест->Enabled = true;
            тестовийПрикладToolStripMenuItem->Enabled = true;
            toolStripButtonРозрахувати->Enabled = true;
            розрахуватиToolStripMenuItem->Enabled = true;
            toolStripButtonОчистити->Enabled = true;
            очиститиToolStripMenuItem->Enabled = true;
            toolStripButtonВідкрити->Enabled = false;
            відкритиФайлToolStripMenuItem->Enabled = false;
      }
      else if (tabControlСторінки->SelectedIndex == 1)//Перевірка чи
активна друга вкладка
      {
            toolStripButtonЗберегти->Enabled = true;
            зберегтиФайлToolStripMenuItem->Enabled = true;
            toolStripButtonТест->Enabled = false;
            тестовийПрикладToolStripMenuItem->Enabled = false;
```

```
            toolStripButtonРозрахувати->Enabled = false;
            розрахуватиToolStripMenuItem->Enabled = false;
            toolStripButtonОчистити->Enabled = true;
            очиститиToolStripMenuItem->Enabled = true;
            toolStripButtonВідкрити->Enabled = true;
            відкритиФайлToolStripMenuItem->Enabled = true;
      }
      else if (tabControlСторінки->SelectedIndex == 2)//Перевірка чи
активна третя вкладка
      {
            toolStripButtonЗберегти->Enabled = true;
            зберегтиФайлToolStripMenuItem->Enabled = true;
            toolStripButtonТест->Enabled = false;
            тестовийПрикладToolStripMenuItem->Enabled = false;
            toolStripButtonРозрахувати->Enabled = false;
            розрахуватиToolStripMenuItem->Enabled = false;
            toolStripButtonОчистити->Enabled = false;
            очиститиToolStripMenuItem->Enabled = false;
            toolStripButtonВідкрити->Enabled = true;
            відкритиФайлToolStripMenuItem->Enabled = true;
      }
      else//Якщо активна четверта вкладка
      {
            toolStripButtonЗберегти->Enabled = false;
            зберегтиФайлToolStripMenuItem->Enabled = false;
            toolStripButtonТест->Enabled = false;
            тестовийПрикладToolStripMenuItem->Enabled = false;
            toolStripButtonРозрахувати->Enabled = false;
            розрахуватиToolStripMenuItem->Enabled = false;
            toolStripButtonОчистити->Enabled = false;
            очиститиToolStripMenuItem->Enabled = false;
            toolStripButtonВідкрити->Enabled = false;
            відкритиФайлToolStripMenuItem->Enabled = false; }
}

private: System::Void MyForm_Load(System::Object^  sender,
System::EventArgs^  e) {
      tabControlСторінки_SelectedIndexChanged(this, e);
}

};

}
```